Easily Testable PLA-Based Finite State Machines

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

Master of Technology

by

Prakash B.

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

FEBRUARY, 1992

Acknowledgment

It gives me great pleasure to express my deep gratitude to Prof. M. M. Hasan for the information, guidance and constant encouragement provided by him during the course of this project. This thesis would not have been possible but for his constant support, along with a personal care that he took at every stage of the work.

I am thankful to Shreesh Yadav, Research Scholar, CS Dept., IITK for the stimulating discussions that we had on many facets of this thesis-topic. My thanks are also extended to my colleagues, P. J. Antony and V. S. Kalari, for the help offered by them during the project. I acknowledge with thanks the many invaluable comments and suggestions given by Ruchir Puri and Raman in doing this thesis.

I was fortunate in having close friends like Waste, Danikkutty, Pillachan, Thadi, Dubey, Chavali, Thandans (I, II), Shankaramman, Koshi, Salim, Annan, Nagpal and Sattan. I would like to thank all of them for their help and support without which the life in IIT campus would have become too dull. I am also indebted to my colleagues P. Gopathy, T. Srikanth, R.S. Anand, D. Ramanath and Vinay Kumar Choudury for their nice company.

I especially wish to thank Gracious, Perappan and their families for the love and affection they poured on me.

Finally, I would like to express my deepest sense of gratitude to my mother and father for the many years of constant encouragement and moral support.

Abstract

The present work addresses the problems of synthesis and testing of *easily testable* PLA-based Finite State Machines (FSM). A new system, named FISTEM, for testability synthesis and test generation of PLA-based FSMs is proposed.

A nonscan design methodology, based on *constrained state assignment and logic optimization*, is used which guarantees testability for all combinationally irredundant crosspoint faults in the PLA. State assignment technique used, which was originally proposed for multilevel logic implementation, is shown to be effective in minimizing the number of product terms in an optimized PLA implementation.

Effect of single crosspoint defects in programmable arrays on their input-output relations has been studied and the necessary constraints required for testability synthesis of a PLA-based Moore or Mealy finite state machine have been formulated. FISTEM makes use of the logic minimization program ESPRESSO to obtain optimized, prime and irredundant two-level covers for output logic (OL) and next state logic (NSL) functions of the FSMs.

Test sequences for all the testable crosspoint faults are obtained using combinational test generating techniques alone. An exact algorithm is used for this to obtain maximum fault coverage. Some powerful heuristics in the area of back end fault simulation and "don't-care" bit fixing are also used in the program, for test set minimality.

For most of the state machine examples considered, the system works very efficiently to produce an optimized easily testable PLA-based logic implementation and a compact set of test sequences to cover all the irredundant faults in the PLA. These results show that the area overhead in return for easy testability is small.

EE-1992-M-PRA-EAS

113540

Table of Contents

Title Page No.

List of Figures

List of Tables

Chapter 1

Introduction

The present work addresses the problems of synthesis and testing of easily testable PLA-based finite state machines. A new system, named FISTEM, for testability synthesis and test generation of PLA based FSMs is proposed. An attempt has been made to use logic synthesis for achieving testability for all the testable crosspoint faults in the in the PLA implementations of FSMs. A review of the available literature, and a brief introduction to the present approach follow in the subsequent sections of this chapter. An overview of the system developed and the organization of the thesis are also included.

1.1 Introduction

The introduction of LSI/VLSI technology has made it possible to place large and complex circuits in a single chip. This in turn has substantially increased the difficulty of both designing and testing digital circuits. The ever increasing complexity of circuits and systems require new and better design methodologies, more powerful CAD tools for realization and verification, and new approaches to the growing problem of testing.

As the number of gates per ICs increase steadily, the number of pins per gate is following an opposite trend. The consequence is an increase in the logic and sequential depths, causing the cost and time spend on testing to rise exponentially. This makes it necessary to consider the subsequent testing problem, in the development stage itself. Given the fact that future electronic sys-

tems will rely heavily on CAD tools, it is clearly important that the testability factors be integrated into these CAD tools. Such an approach has been adopted in FISTEM, which uses a procedure of constrained state assignment and logic optimization to obtain easy testability for PLA-based FSMs.

PLA macros have gained tremendous popularity in complex VLSI system design owing to their topologically regular structure and to the efficient software tools (*e.g.*, [18], [26]) which are now available for two stage logic implementation and test generation. The programmable nature of the PLA makes the design task much easier and its array oriented structure simplifies the testing problem. This approach exploits many of the benefits of LSI/VLSI without the high engineering design costs. Finite state machines play a major role in the control part of digital systems and it can be realized very efficiently by adding feedback registers to PLAs. Here the PLA is two-level combinational logic implementation of the output logic (OL) and the next state logic (NSL) functions of the FSM. The general structure of PLA-based finite state machine is shown in Figure 1.1. Direct access is provided only to the primary inputs (PI) and primary outputs (PO).

Design-for-testability techniques for PLAs require controllability of all inputs and observability of all outputs of the PLA. The present approach to attain easy testability doesn't require direct access to the flip-flops.

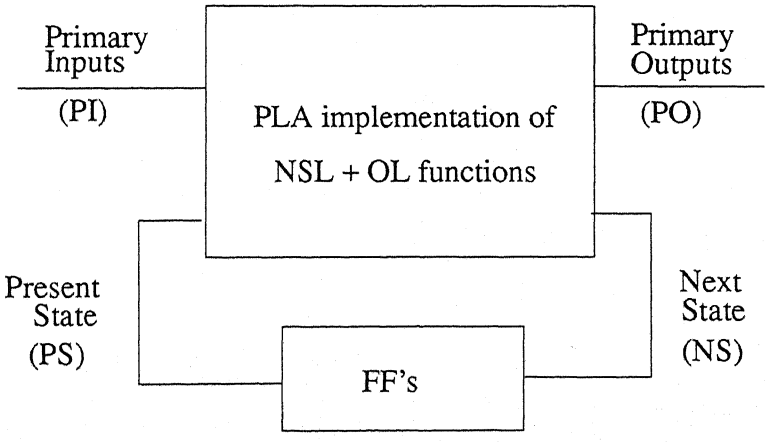

Figure 1.1: PLA-based Finite State Machine

1.2 Literature Review

Synthesis and testing of PLA-based structures have been active areas of research. Numerous programs for optimal synthesis of PLA-based finite state machines have been developed (*e.g.*, [16], [18]). As mentioned in §1.1 the present work deals two major tasks namely

- improving the testability of finite state machines, and
- generating test sequences for them.

Extensive work has been done on each of the above, and considerable literature is therefore available, some of which is detailed below.

1.2.1 Testability of Finite State Machines

FSM testing has long been recognized as a difficult task and much effort has been directed towards this problem in recent years. Design-for-testability (DFT) has been proposed as a solution to this problem [11]. This approach tries to standardize the circuit design in such a way that test preparation for the circuit is made easier. The design rules suggested [22] for improving controllability and observability of the circuits include

- limiting sequential depth in the test mode,
- disconnection of feedback loop in the test mode,
- access to internal data busses via external pins, and
- disconnection of internal stores.

Among various DFT approaches proposed for sequential circuits, the full scan design techniques [27] such as LSSD, Scan Set and Scan Test have become the most popular. These approaches make it possible to control and observe the memory elements via some straightforward mechanism and convert the difficult problem of testing state machines into a much easier one, that of testing combinational circuits. However, the long testing time due to the serial nature of the scan

path, and the area/performance penalty with this approach are unaffordable in many cases [9]. Scan design penalties for different classes of sequential circuits were placed between 4 and 21% [7].

It is well known that optimal logic synthesis can ensure fully testable combinational logic designs. A synthesis procedure which guaranteed fully testable irredundant combinational logic circuits was presented in [2]. In principle, logic synthesis and minimization techniques can ensure full and easy testability for sequential logic also. Relationships between complicated problems of sequential circuit synthesis and test generation has been discussed in [9], [10], [6] and [7].

Major steps involved in optimal synthesis of FSMs from the State Transition Graph (STG) description are:

- State minimization,
- state assignment, and
- logic optimization.

The work presented in [9] showed that state assignment and logic optimization have profound effect on the testability of the state machine. In [9], a procedure which produced fully and easily testable logic-level sequential machine from a STG description was proposed. However, to achieve this goal, this approach has entailed the use of extra logic and constraints on state assignment and logic optimization. Another optimal synthesis procedure that guarantees full nonscan testability under the stuck-at fault model, with no area/performance penalty has been proposed in [10]. Though this method represents a more efficient approach to achieve 100% testability, amounts of CPU time it require is, generally, exorbitant. A unification of these synthesis-for-testability approaches aimed at stuck-at faults was presented in [6].

However, the single stuck-at fault model is inadequate to represent the faults in PLAs. According to [26], the stuck-at oriented test sets are less reliable and will often not reach 90% coverage for faults in programmable arrays. Due to PLA's dense layout, PLA faults other than conventional stuck-at faults can occur easily and must be modelled. The work in [14] showed that test sets based on crosspoint fault model, an extended model proposed in it, covers many of the frequently occurring physical faults in the PLA, including shorts between lines. A method to produce easily

testable PLA-based state machines, without requiring direct access to the inputs/outputs of the circuit's memory elements, has been outlined in [7]. This method aims at the crosspoint fault model and guarantees testability for all combinationally irredundant crosspoint faults in the PLA-based finite state machine. A constrained state assignment and logic optimization procedure is used to achieve this goal. The testability constraints employed in [7], in general, conflict with the optimality constraints derived from the STG and hence result in an increase in area. More over, in some cases state assignment satisfying all the testability constraints can even become impossible. The present work adopts a similar approach, but for a modified set of constraints.

1.2.2 Test Generation

In order to detect a fault in a sequential machine,

1. the state of the memory elements and the input have to be set into a certain combination that can excite the fault and

2. the effect of the fault has to be propagated to the primary outputs.

These steps, fault justification and propagation, may each require a sequence of input vectors (justification and distinguishing sequences).

Several approaches (*e.g.*, [25], [4]) have been taken in the past to solve the problem of test generation for sequential machines. They are either extensions to the classical *D*-algorithm or based on random techniques. Major drawback of these conventional techniques is that they rely on backtracking. For sequential circuits, backtracking must be done both in space and time and could easily become unmanageable [1]. When the number of states of the circuit is large and the tests demand long input sequences, they can be quite ineffective for test generation.

For those state machines, which are designed using DFT techniques, test generation is relatively simple. The structural approaches discussed in §1.2.1 convert the circuit under consideration into combinational one, in the test mode. The state-of-the-art approach is to reduce the problem of test generation for sequential circuits essentially to one of generating test patterns for combinational circuits. Combinational test generation takes a crucial role in the test generation for easily testable sequential machines [7] also. For these machines, a possible test sequence is

obtained by concatenating the justification sequence, the combinational test vector, and the distinguishing sequence.

Many dedicated algorithms [14], [12], [24], [23], [26], [20] for PLA test generation have been proposed. These techniques aim at crosspoint fault model and exploit the regularity of the PLA structure for ease of computation and test set optimality. Automatic test pattern generation techniques, available for random combinational logic circuits are not effective for PLA macros. Some of these methods such as *D*-algorithm and its extensions [4] require a functionally equivalent random logic model of the PLA. If only the personalized crosspoints are considered in this model, other types of faults due to the dense layout of the circuit may be undetected by the test set [14]. When a crosspoint is redundant, the process of fault sensitizing and consistency operations must exhaust all locally viable signal values. This presents an enormous computational burden in conventional test generation methods.

Random techniques do not give high test coverage for PLAs mainly because the AND array in PLA normally has a relatively large number of "*used crosspoints*" in each product term [12]. The probability of detecting a missing crosspoint with a random pattern is no better than $1/2^n$, where n is the number of used crosspoints in the product term. Since n is very large, the test coverage using random test is quite low. In [12], a biased random technique has been used for test generation. In this approach, not all the bits are randomly assigned; instead, deterministically formed embryonic tests are used as starting point, and all the unspecified inputs are selected randomly.

The basic approach adopted in [14], [23], [24], [26] and [20] is same and they all require the use of sharp operation for the deterministic test generation. Major differences among these algorithms lie in the way the *sharp* operation is carried out and how heavily this operation is depended on. In [14], the sharp operation is performed by IDSHARP ($). In [23] and [20], regular sharp(#) is used; but the computation is stopped as soon as one face of the set difference is found. In [24], a list partitioning method is proposed. A new algorithm based on complementation and tautology check [3] of a logic cover derived from the PLA personality matrix has been used in PLATYPUS[26]. Test generation program presented in [20] makes use of deductive fault simulation method, proposed in [14], for each test vector found using deterministic method, and discard the faults covered by that pattern from the faultlist. This improves the total run time performance as

well as the test set minimality.

1.3 Present Work

The present work proposes a new CAD system, named FISTEM, for synthesis and testing of FInite State - easily TEstable Machines. FISTEM produces an optimized easily testable PLA-based logic implementation for the input state transition graph (STG) description of a sequential machine. A nonscan design methodology, based on *constrained state assignment and logic optimization*, is used which guarantees testability for all combinationally irredundant crosspoint faults in the PLA. Test sequences for these faults have been obtained using combinational test pattern generation techniques alone. Algorithm used for this is exact, *i.e.*, every testable crosspoint fault is tested, and maximum fault coverage is *guaranteed.*

The problem of constrained state assignment is to find an encoding of states which minimizes the combinational logic part of the sequential machine simultaneously satisfying all the testability constraints imposed. The state assignment technique used in FISTEM has been originally proposed in [8] for multilevel combinational logic and feedback register implementation. Both fanin and fanout oriented algorithms proposed in [8] are implemented. The results obtained for various STGs show that this technique is also effective in minimizing the number of product terms in an optimized PLA implementation. Also an attempt is made to combine the fanin and fanout oriented algorithms.

Effect of single crosspoint faults in programmable arrays on their input-output relations has been studied. Relationship between state assignment and testability has also been analyzed, and the necessary constraints required for testability synthesis of a PLA-based Moore or Mealy finite state machine have been formulated.

Graph embedding algorithm of MUSTANG [8] is used for optimum encoding of the state, satisfying all the testability constraints. Once the code has been assigned, product terms of the PLA can easily be obtained by substituting the present and next states by their corresponding codes in each row of the State Transition Table (STT). Logic minimization program ESPRESSO [3] is used for obtaining optimized prime and irredundant two-level covers for the output logic (OL) and next state logic (NSL) functions of the easily testable FSM. For the sake of simplicity, *D*-flip-

flops are considered for memory elements of the FSM.

Algorithm proposed in [20] is used for sequential test generation. Many heuristics in the area of global ordering of the faults for test preparation, backend fault simulation, and " don't-care" bit fixing are also incorporated in the program for obtaining the maximum fault coverage and test set minimality. All the crosspoint faults in the PLA are considered. However, testing of internal faults of the memory elements is not attempted. A starting or initial state is assumed to exist for a machine, also called the reset state.

1.4 System Overview

FISTEM is a system for FSM synthesis and testing, intended to be properly interfaced with other existing tools to generate a comprehensive design environment. During system development, three goals were constantly pursued:

- Complete integration with the existing CAD system,
- full portability, and
- friendly interface.

To satisfy the first aspect, an attempt is made to achieve complete compatibility with already existing CAD tools for PLAs and PLA-based FSMs.

Full portability of the overall system has been obtained both by implementing it in standard C and by pursuing maximum independency from the operating system of the host computer.

In order to guarantee a flexible environment, a modular structure is used for the system. Functional descriptions of the each module, elucidated in Figure 1.2, are presented below.

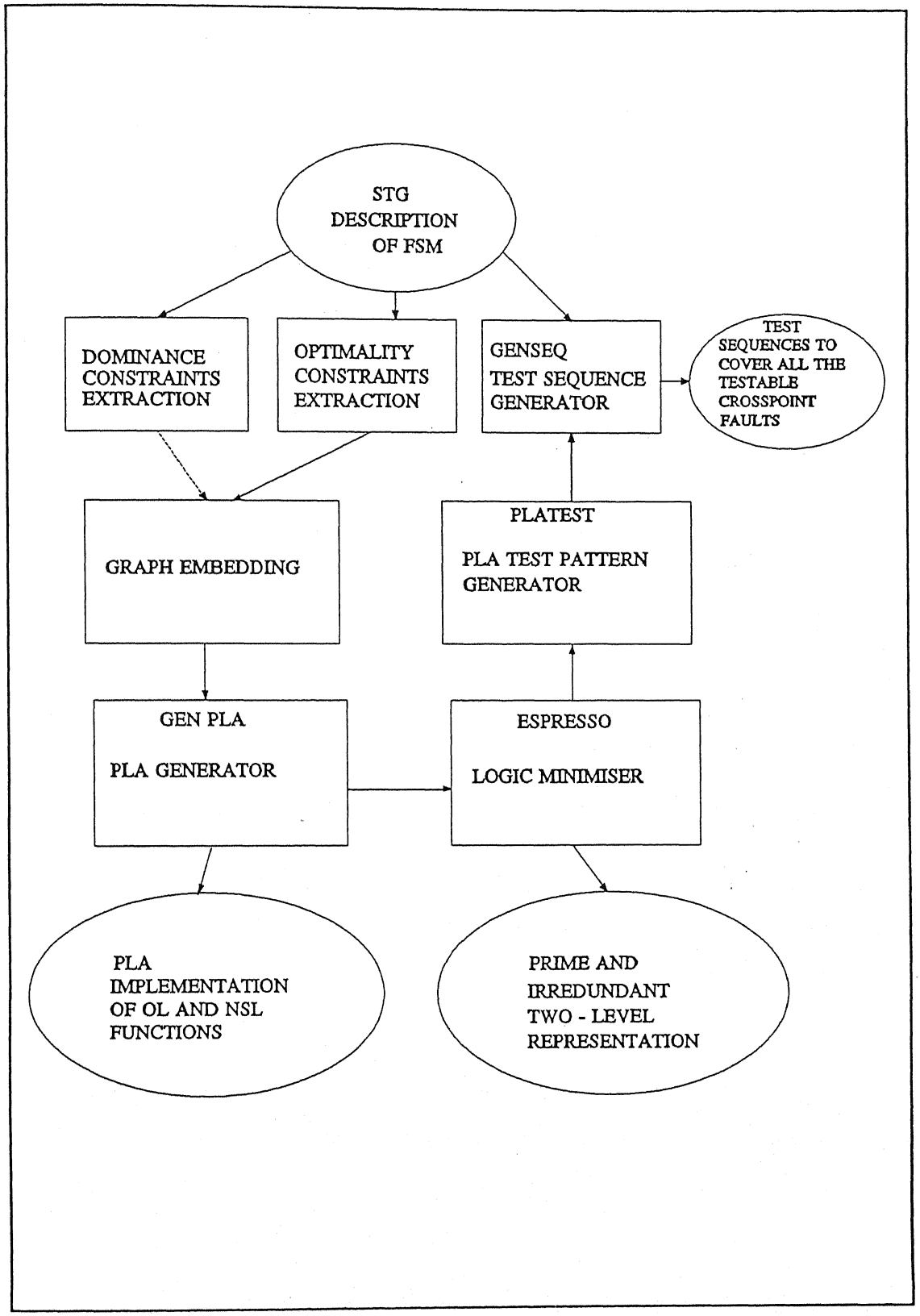

Figure 1.2: System Overview

Dominance constraints extraction: Adds edges to the input STG so that R-reachability condition is satisfied. Extracts the sets of dominance relationships between each pair of states which has to be satisfied for ensuring easy testability.

Optimality constraints extraction: From the STG of the R-reachable finite state machine, this module constructs a completely weighted graph, whose nodes correspond to the internal states of the state machine to be encoded. The weights of the edges signify the gains associated with coding the pair of states connected by the edges with unidistant codes. Users can choose between Two algorithms implemented for this step: Fanin oriented and fanout oriented algorithms.

Graph embedding: This subroutine embeds the weighted graph, constructed in the previous step, in Boolean hypercube so that adjacency relations identified by the graph are satisfied in an optimal way. A heuristic graph embedding algorithm called *wedge clustering* [8] is used to solve this *NP*-complete problem. The originally proposed approach is modified to incorporate testability constraints extracted in step 1, in the state encoding. When a node is selected for coding, all the dominance requirements associated with that node are examined, and the optimality constraints are relaxed, if necessary, to satisfy them.

GENPLA: GENPLA generates two-level PLA implementation for the finite state machine represented in the input STG. The present and next states in each row of the state transition table are replaced by their corresponding codes to obtain the two-level representation of OL and NSL functions of the FSM.

ESPRESSO: The Boolean minimizer program ESPRESSO [3] produces an optimized implementation of the functions represented in the output of GENPLA. It assures prime and irredundant covers for these functions, which in turn will guarantee testability for all the stuck-at faults in the PLA.

PLATEST: This module accepts the PLA personality matrix as its input and generates a compact set of test patterns to cover all the combinationally irredundant crosspoint faults in the PLA.

GENSEQ: This program uses the list of test pattern generated by PLATEST for finding the sequences. For an easily testable machine, GENSEQ generates a set test sequences, which will cover all the crosspoint faults covered by the input test patterns. Test sequences are obtained by concatenating a fault-free justification sequence, the input test vector and a distinguishing vector (which is guaranteed to exist for an easily testable FSM).

FISTEM is implemented in the UNIX environment of HP 9000 Series 300 Graphic Workstations.

1.5 Organization of the Thesis

The organization of this write-up is as follows. In Chapter 2, the state assignment technique used in FISTEM is described. Effect of crosspoint faults on the input-output relations of the PLA is discussed in Chapter 3. The relationship between the state assignment and the testability of sequential machine is also discussed, and the necessary conditions required for an easily testable PLA-based finite state machine are stated. This Chapter also outlines the constrained encoding procedure, used in the present approach for easy testability. In Chapter 4, combinational and sequential test generation techniques used in this utility are covered. The results obtained thus far are discussed in Chapter 5.

Chapter 2

State Assignment

The optimal state encoding approach followed in FISTEM is discussed in this Chapter, and its efficacy is illustrated with a STG example taken from the MCNC Logic Synthesis Workshop (1987) benchmark set. In the following section, some notations and terminologies used in the area of FSM synthesis, in particular state assignment, are introduced. §2.2 describes the two basic processes behind the influence of state encoding on the subsequent Boolean minimization step. In §2.3, Graph Construction and Embedding algorithms used for optimizing the product of row and column cardinalities of the PLA implementation are outlined. These techniques were originally proposed in MUSTANG [8] for multilevel logic implementation. Attempts for additional reduction in logic are covered in the subsequent section.

2.1 Background

Sequential circuits play a major role in the control part of most of the complex VLSI systems. Finite state machine is one of the most commonly used models for representing a sequential machine. Assigning states for the memory elements corresponding to the internal states of the finite automation is a major step in the process of controller synthesis. Expectedly, the quality of the encoding has a significant impact on the overall chip area. Before discussing the state assignment problem of FSMs, it becomes essential to define a few basic terms which are relevant to the present work.

2.1.1 Definitions and Terminologies

A finite state machine can be defined as a quintuple

$$M = (S, I, O, \delta, \lambda)$$

where

(i) S is a finite nonempty set of states;

(ii) I is a finite nonempty set of primary inputs (PI);

(iii) O is a finite nonempty set of primary outputs (PO);

(iv) δ: $I \times S \rightarrow S$ is the next state logic (NSL) function; and

(v) λ: $I \times S \rightarrow O$ is the output logic (OL) function.

The above definition is for a *Mealy* FSM in which the PO's are a function of both the present state and the PI's. Mathematically speaking, a *Moore* machine is a special case of Mealy machine and for it, the same definition holds good except for the part (v). For Moore machine, the OL function is

$$\lambda: S \rightarrow O.$$

In this work, PLA-based Mealy machines are considered, since a Mealy machine can be viewed as a more general case of a Moore machine.

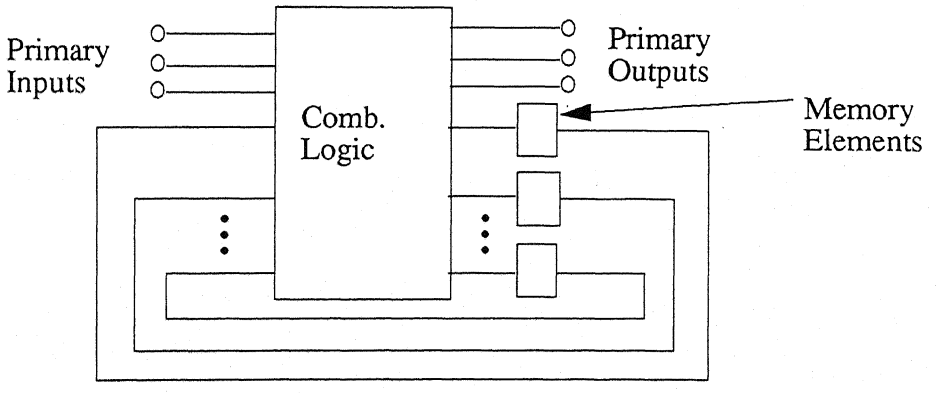

Figure 2.1: Basic Structure of the Huffman Model

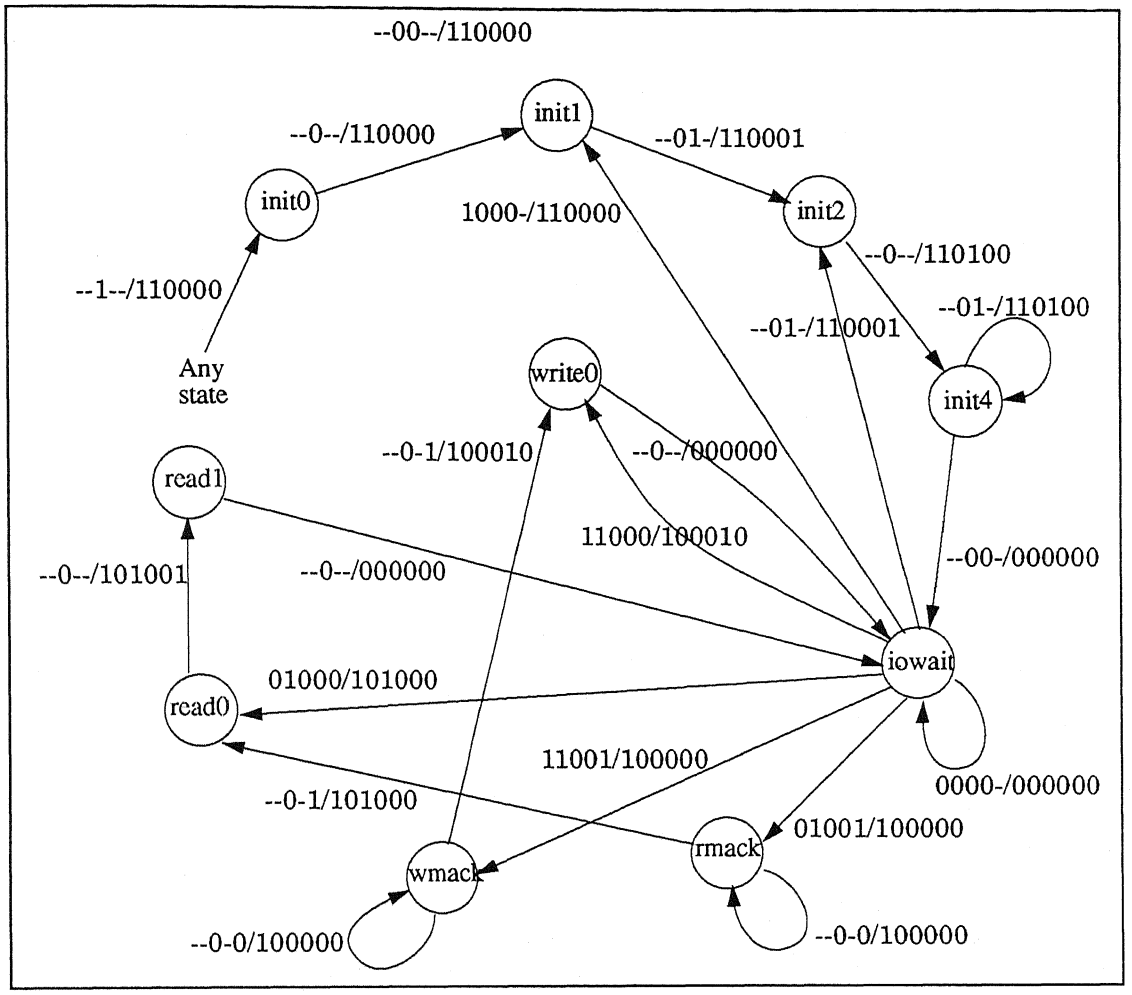

Figure 2.2: State Transition Table Representation of IOFSM

Basic structure of the Huffman model is shown in Figure 2.1. This model is one of a sequential machine that separates the memory elements from the combinational logic. In general, an FSM is represented by two equivalent structures.

1) State Transition Graph (STG): $G(V, E, W(E))$ where V is the set of vertices corresponding to the set of states S, where $\| S \| = N_s$ is the cardinality of the set states of the FSM, an edge (V_i, V_j) joins V to V_j if there is a primary input that causes the machine to evolve from state V_i to state V_j, and $W(E)$ is a set of labels attached to each edge, each label carrying the information of the primary inputs and outputs, corresponding to that transition.

2)State Transition Table (STT): $T(I, S, O)$ where I is the set of inputs, S is the set of states, and O is the set of outputs. A row in the STT corresponds to an edge in the STG representation. The rows in the matrix are divided into two fields: The first field contains the input pattern (In this work, it is assumed that the primary inputs and outputs are in the Boolean form.) and a symbolic entry corresponding to the present state; the second field contains the output pattern and the next state labels.

```
Inputs: IO1, IO0, INIT, SWR, MACK
Outputs: WAIT, MINIT, MRD, SACK, MWR, DLI

--1--   Any     init0   110000
--0--   init0   init1   110000
--00-   init1   init1   110000
--01-   init1   init2   110001
--0--   init2   init4   110100
--01-   init4   init4   110100
--00-   init4   iowait  000000
0000-   iowait  iowait  000000
1000-   iowait  init1   110000
01000   iowait  read0   101000
11000   iowait  write0  100010
01001   iowait  rmack   100000
11001   iowait  wmack   100000
--01-   iowait  init2   110001
--0-0   rmack   rmack   100000
--0-1   rmack   read0   101000
--0-0   wmack   wmack   100000
--0-1   wmack   write0  100010
--0--   read0   read1   101001
--0--   read1   iowait  000000
--0--   write0  iowait  000000
```

Table 2.1: State Transition Table representation of IOFSM

If the next state and outputs are defined for every possible transition from every state, then the machine is said to be a *completely specified machine*. An *incompletely specified machine* is one where at least one transition edge from some state is not specified.

Given n latches in a sequential machine, 2^n possible states exists in the machine. A starting or initial state is assumed to exist for all machines, also called the *reset state*.

2.1.2 Optimal State Encoding

The state assignment problem consists of assigning a string of bits (code) to each of the states, so that no two states have the same code. Depending on the target architecture, the encoding problem can be quite diversified. For a microprogrammed controller, the problem consists essentially of organizing the control memory; a state encoder in this context performs the synthesis of the microsequencer. In the case of synchronous finite state machines based on combinational logic and feedback registers, state assignment aims at minimizing the next state and output equations. The optimal encoding, in this case, is the one which yields the minimum area and delay, in the final implementation.

The complexity of the combinational component of the FSM depends heavily on the state assignment and the selection of memory elements. D flip-flops are considered in the present work, for the sake of simplicity. For a PLA-based FSM, the (minimum) number of rows is the cardinality of the (minimum) cover of the FSM combinational component according to a given assignment. The number of bits used to represent the states is related to the number of PLA columns. Therefore, the PLA area depends in a complex way on the state assignment. However, in FISTEM, the attempt made is to minimize the number of product lines in the PLA, keeping the number of encoding bits minimum ($n = \lceil \log_2 N_s \rceil$).

The assignment approach proposed in [8] minimizes an estimate (the number of literals in a factored form of the logic) of the area used by a multilevel implementation of the combinational logic. This estimate is consistent with the one used by the multi-level logic optimization algorithms. Basically, the heuristics used in this approach tries to maximize the number and size of the common subexpression that exist in the Boolean representation of the combinational part of the FSM. Presence of these common cubes helps the subsequent multi-level optimization step to extract a large number of good factors and produce a reduced literal implementation.

In the present work, it is found that maximizing the number and size of the common cubes present in the encoded two-level network can also aid the two-level logic optimizer (ESPRESSO) in minimizing the next state and output equations. But, in terms of implementations of minimum area PLA, this approach suffers from a weak connection with the two-level logic optimization step, which follows the encoding. It is the simplicity of the concept and its suitability for incorpo-

rating the testability constraints (discussed in Chapter 3) which makes this method suitable for the present work. Some improvement is also tried on this technique to obtain better results for PLA implementations. Once the encoding has been done, the codes assigned to each state is shifted in the Boolean hypercube (keeping the hamming distance same) so that the state with maximum fanin will be assigned zero-code. It has been found that for most of the STG examples considered, this modification produces a further reduction in the number of product terms. In the following sections, basic idea behind this approach is discussed and the major steps involved in it are illustrated with the IOFSM example shown in Figure 2.2.

2.2 Relationship between State Encoding and The Number/Size of Common Cubes

Presence of common cubes in the encoded state transition table can help both two-level and multi-level optimizations in obtaining a reduced area implementation. There are two basic processes behind the influence of state assignment on the number or size of common cubes in the encoded two-level representation. These two processes are analyzed below.

2.2.1 Size of The Frequently Occurring Common Cubes

It can be noticed from the STT of IOFSM (Table 2.1) that the present states which assert similar outputs or produce similar sets of next states can give rise to common cubes after encoding. The size of these common cubes depend on the *closeness* of the codes for these present states.

Focussing on edges 2 and 3 in Table 2.1, it can be seen that both the states `init0` and `init1` assert the outputs O1 and O2. For each of these outputs it is possible to extract a common cube corresponding to the intersection of the two state codes. In addition to this, both these states produce the same next state `init1`. So the lines of next state `init1` also will have the same common cube. Similar relationships exist between other sets of states in the machine.

If N_b is the number of encoding bits and N_d is the hamming distance between codes for the present states `init0` and `init1`, then the lines of next state `init1` and the outputs O1 and O2 will have common cube with N_b - N_d literals, *i.e.*, closer the codes for `init0` and `init1`, larger

will be the size of the common cube associated with them.

For a primary output asserted by two different present states, number of occurrences of the corresponding common cubes is one. But for the next state produced by a pair of present states, the number of occurrences of these cubes is dependent on the encoding of that next state. Since the number of 1's in the next state code is unknown prior to encoding, it has to be approximated to an average number $N_b/2$. Using this average case analysis, one can very easily estimate the number of occurrences of the common cube associated with any pair of states in the machine.

For a machine, thus a large set of relationships between state encoding and the size of common cubes in the network prior to logic optimization, can be extracted. The number of occurrences of a common cube associated with a state pair can be considered as the *gain* that can be obtained by coding that pair with close codes. To maximize the overall gain, closer codes should be given to state pairs with higher values of gain. This process, in turn, will maximize the size of the most frequently occurring cubes.

2.2.2 Number of Common Cubes in The Next State Lines

This process is based on the relationship between the next states which are produced by similar primary inputs or similar sets of present states. In this case, the process is shifted to the input and fanin of each state. In edges 9 and 10 of the STT representation of the IOFSM (Table 2.1), the present state `iowait` produces different next states *viz* `init1` and `read0`. If N_d is the hamming distance between the codes for these next states and N_b is the number of encoding bits, a maximum of N_b - N_d next state lines can have a common cube (the present state code) of size N_b literals. In fact, the number of occurrences of this present state common cube depends on the number of 1's in the intersection of the two next state codes. Similarly, for these edges, intersection of the input combinations `1000-` and `01000` will be common to some of the next state lines. The size of this common cube can be easily found out for each pair of next states. Again, similar relationships exist between other sets of states in the FSM.

The size of the common cubes corresponding to each pair of next states can be considered as the gain that can be achieved by assigning close codes for these states. *i.e.*, Larger the size of the common cubes for a next states pair, more will be the gain in assigning close codes for them. In

general, this approach will attempt to maximize the number of occurrences of the largest common cubes in the encoded machine, prior to optimization.

2.3 MUSTANG Algorithms

The algorithms used in the state assignment program MUSTANG, targeted towards multi-level implementation are based on the two basic processes discussed in the previous section. In general, the constraints extracted for optimum state encoding on the basis of these two processes contradict each other. Therefore, two independent algorithms are used in this approach to construct a fully connected graph which will provide the costs of assignments of internal states of FSM to the vertices of the Boolean hypercube. In the following subsections, these graph construction algorithms, called fanin and fanout algorithms, and the algorithm used for graph embedding are discussed in detail. A heuristic algorithm called *wedge clustering* is used for the graph embedding step. This algorithm assigns codes to the nodes of the constructed graph, satisfying the adjacency relations identified by it in an optimal way. FISTEM makes use of these algorithms for obtaining an optimum two-level implementation of the input STG.

2.3.1 Fanout-Oriented Algorithm

This algorithm works on the output and fanout of each state in the finite state machine and constructs a complete graph $G_M(V, E_M, W(E_M))$ where V is in one-to-one correspondence with the states of the FSM, E_M is the complete set of edges, and $W(E_M)$ represents the gains that can be achieved by coding the states joined by the corresponding edges as close as possible. The estimation of these gains are based on the analysis of §2.2.1. Major steps involved in the construction of the graph are:

1. For each of the N_o outputs, scan all the edges of the STG to identify the nodes which assert that output, and calculate the weight associated with each node. If a state assets the same outputs more than once, it should have a correspondingly larger weight.

2. From these output sets, calculate the number of occurrences of common cubes corresponding to each pair of states. This can be estimated by multiplying the weight of two nodes corresponding to the state pair across all the sets.

3. For each of N_s next states, construct a set of weighted nodes/states which produce that next state, by scanning all the edges of the STG.

4. For each state pair, calculate the occurrences of the corresponding common cubes in the next state field. This is, again, equal to the product of corresponding node-weights across all the next state sets. As discussed in §2.2.1., an average case analysis is carried out in this case, and the above mentioned product is multiplied by $N_b/2$.

5. Construct the undirected complete graph $G_M(V, E_M, W(E_M))$. Calculate the edge weight $W(E_M)$ as the sum of the number of occurrences of common cubes estimated in step 2 and 4.

These weights in the constructed graph G_M represent a set of closeness criteria for the states in the machine and will help the graph embedding step to maximize the size of the most frequently occurring cubes.

2.3.2 Fanin-Oriented Algorithm

This algorithm is based on the analysis of §2.2.2. and operates on the input and fanin of each state in the machine for calculating the edge weights. The algorithm proceeds as follows:

1. For each of the N_i inputs, scan all the edges in the STG to identify the sets of next states which are produced by input value 1 and value 0. This will give $2N_i$ sets of weighted nodes.

2. Count the number of times each pair of states occurs together in each input set. This step, in effect, will compute the similarity relationships between all the next state pairs in terms of inputs. It can be noticed that the weight calculated for each pair of states represents the size of the common input cubes in the next state lines.

3. For each present state, construct a weighted set of next states which fanout from that present state. This step should give N_s weighted sets.

4. From the next state sets, compute edge weights for each pair of states. To obtain this, multiply the weights of the two corresponding nodes across all the sets. This will give next state pairs which have many common present states correspondingly higher edge weights. This step is similar to the approach used for input sets, except for an additional multiplication factor N_b. Since each of the common present state cubes have N_b literals as opposed to the single literal cube of primary inputs, the weights are multiplied by N_b in this step.

5. Combine the weights computed in step 2 and 4. In this algorithm, these edge weights will represent the sizes of different sets of common cubes in the machine. By assigning closer codes for states joined by edges of larger weights, the number of occurrences of common cubes can be maximized.

2.3.3 Graph Embedding

Graph embedding is a classical combinatorial optimization problem which effectively maps the weighted graph G_M, produced by the fanin/fanout algorithm to the Boolean hypercube so that the adjacency relations identified by the graph are satisfied in an optimal way. In fact, this problem is an *NP*-complete one and there is very little hope to solve it exactly in an efficient way. In the present case, a heuristic algorithm proposed in [8], called *wedge clustering* is used to assign codes to the internal states of the machine according to the analysis performed by fanin and fanout algorithms. This algorithm attempts to minimize

$$\sum_{i=1}^{N_s} \sum_{j=i+1}^{N_s} Wt(E(v_i, v_j)) \times HD(code(v_i), code(v_j))$$

where $Wt(\ E_M(\ v_i, v_j\ ))$ is the weight of the edge $E(\ v_i, v_j\ )$ between vertices v_i and v_j, and $HD(\ code(\ v_i\ ), code(\ v_j\ ))$ is the hamming distance between Boolean codes assigned to the vertices v_i and v_j.

The weighted graph generated by the fanin and fanout algorithms typically contains strongly connected small group of states. The wedge clustering approach tries to identify these strongly connected clusters in the graph and assigns minimally distant codes to the states in it. A detailed

discussion on the optimality of this approach can be found in [8]. The algorithm proceeds as follows:

1. For each state in the FSM, sort the N_s -1 edges connected to that state in decreasing order of their edge weights.

2. Identify a cluster of nodes with cardinality $N_b + 1$, so that the sum of edge weights inside the selected cluster is maximum.This can be obtained by identifying the state for which the sum of weights of the first N_b edges in the sorted list is maximum. This state can be considered as the head of the selected cluster.

3. Assign all the unassigned states in the cluster, minimally distant codes from unassigned codes. An easy and satisfactory way to do this is to assign unidistant codes from the head node, for rest of the nodes.

4. Delete the head node/state and all the edges connected to it from the graph.

5. If the graph is not empty, go to the step 2.

In the following section, the STG example shown in Figure 2.2 is encoded using the graph construction and embedding algorithms discussed above. The results obtained illustrate the efficacy of this approach in optimizing the number of product terms in PLA-based FSMs. Both the graph construction algorithms implemented in FISTEM are very much similar except for the field in STG, on which it operates to estimate the edge weights. Hence, only the fanin-oriented algorithm is considered in the following example.

2.3.4 Example

The input STG considered in this example is IOFSM, a state machine taken from the MCNC Logic Synthesis Wokshop (1987) bench mark set. The STG and STT representations of this FSM are shown in Figure 2.2 and Table 2.1 respectively. The major steps in both graph construction and embedding are illustrated below.

Weighted Graph Constuction: The input STG, IOFSM, has got 5 primary inputs and 10 internal states. For each of the inputs, two sets of weighted nodes are constructed: one for zero value and the other for 1. Zero-input sets and One-input sets are seperately given in Table 2.2 and 2.3 respectively.

state / input	init0	init1	init2	init4	iowait	rmack	wmack	read0	read1	write0
IO1	0	1	0	0	0	0	1	0	0	1
IO0	0	0	0	0	0	1	1	1	0	1
INIT	1	0	0	0	0	0	0	0	0	0
SWR	0	0	2	1	0	0	0	0	0	0
MACK	0	0	0	0	0	1	1	1	0	1

Table 2.2: Zero-input sets for IOFSM

state / input	init0	init1	init2	init4	iowait	rmack	wmack	read0	read1	write0
IO1	0	0	0	0	1	1	0	1	0	0
IO0	0	1	0	0	1	0	0	0	0	0
INIT	0	3	2	2	4	2	2	2	1	2
SWR	0	2	0	0	2	1	1	1	0	1
MACK	0	0	0	0	0	1	1	1	0	1

Table 2.3: One-input sets for IOFSM

In each set, the weight of a node is the number of times the corresponding state appears in the next state field of STT, for that input value. For example, there are two transitions in the STT (Table 2.1) in which a zero-value for input `SWR` produces a next state `init2` and hence the entry 2 in Table 2.2. Similarly there are four transitions in IOFSM for which input `INIT` is 1 and the next state is `iowait`. Note that these entries are high lighted in Table 2.2 and 2.3.

Table 2.4 shows the present state sets generated for IOFSM. In a present state set, the weight of a node/state is the number of transitions in the STT in which this state and the present state under consideration appear as NS-PS pairs. In the example considered, there is no transition from `iowait` to `init4` and hence the weight of `init4` in `iowait`-set is 0.

	init0	init1	init2	init4	iowait	rmack	wmack	read0	read1	write0
init0	1	1	0	0	0	0	0	0	0	0
init1	1	1	1	0	0	0	0	0	0	0
init2	1	0	0	1	0	0	0	0	0	0
init4	1	0	0	1	1	0	0	0	0	0
iowait	1	1	1	0	1	1	1	1	0	1
rmack	1	0	0	0	0	1	0	1	0	0
wmack	1	0	0	0	0	0	1	0	0	1
read0	1	0	0	0	0	0	0	0	1	0
read1	1	0	0	0	1	0	0	0	0	0
write0	1	0	0	0	1	0	0	0	0	0

Table 2.4: Present state sets for IOFSM

Using these zero-input sets, one-input sets and present state sets a complete, undirected, weighted graph is constructed as shown in Figure 2.3. Weights on the edges between each pair of nodes in the graph is calculated by adding the product of the weights of the two nodes across all the sets. Weights computed from the present state sets are scaled by a factor $N_b = \lceil \log_2 10 \rceil = 4$. For example, weight on the edge between `write0` and `iowait`

$$Wt(E(\texttt{write0, iowait})) = 0\times1+0\times1+0\times0+0\times0+0\times1+1\times0+1\times0+4\times2+2\times1+0\times1+ 4(0\times0+0\times0+0\times0+1\times0+1\times1+0\times0+0\times1+0\times0+1\times0+1\times0) = \underline{14}$$

The size of the common cubes considered in the present state sets is Nb = 4, where as in input sets common literals are under consideration. Therefore the contribution of the present state sets to the edge weights are multiplied by 4 so that the weights are true representation of the size of the common cubes.

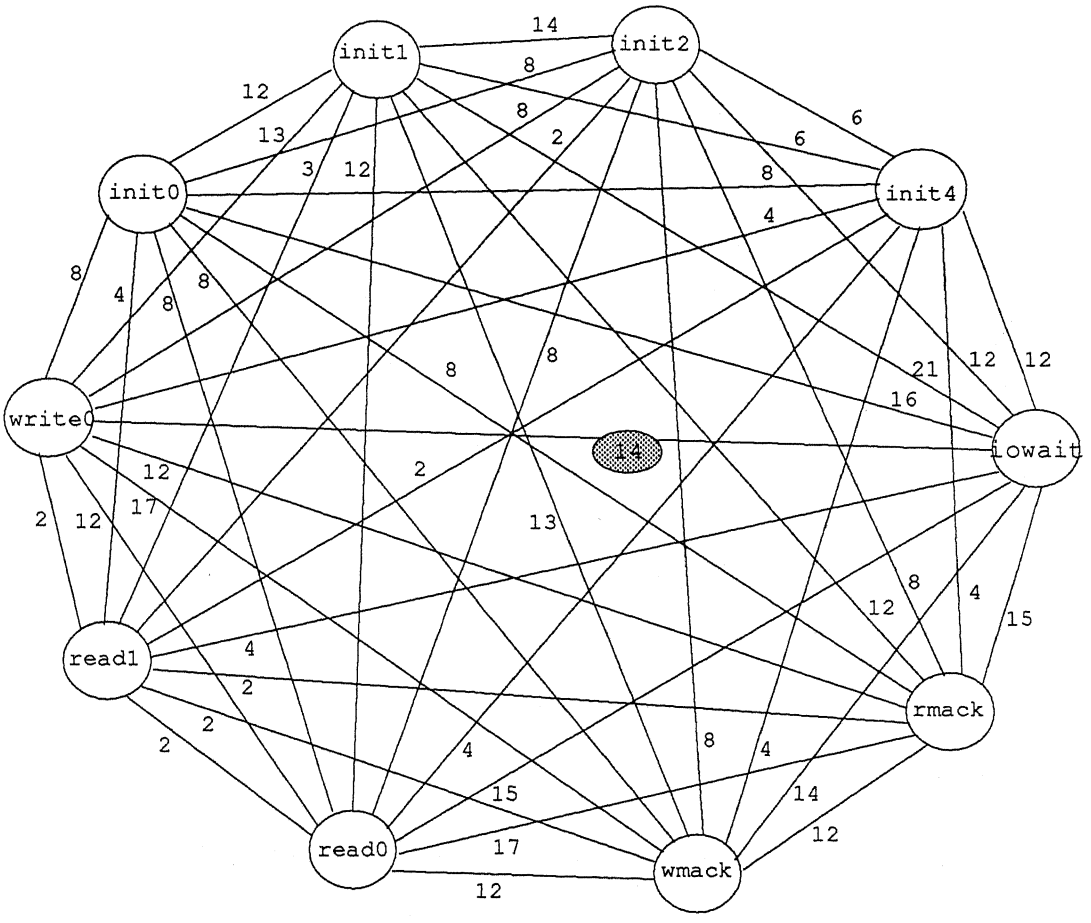

Figure 2.3: Weighted graph generated by the Fanin algorithm

Graph Embedding: For each state, edges are sorted in the decreasing order of weights as shown in Table 2.5. The next step is to select a strongly connected cluster of 5 states. It can be seen that iowait is the state for which the sum of weights of the first 4 edges is maximum. So the cluster selected includes `iowait`, `init1`, `init0`, `rmack`, and `read0`. Here, states `init1`, `init0`, `rmack`, and `read0` are assigned unidistant codes from `iowait`.

`iowait` → 0000 `init1` → 0001 `init0` → 0010 `rmack` → 0100 `read0` → 1000

Selection of code 0000 for state `iowait` is an arbitrary one. Now `iowait`, the head-node of the cluster and all the edges connected to it are deleted from the graph. Sorted lists are updated accordingly (Table 2.6).

init0	iowait (16)	init1 (12)	init2 (8)	init4 (8)	rmack (8)	read0 (8)	wmack (8)	write0 (8)	read1 (4)
init1	iowait (21)	init2 (14)	write0 (13)	wmack (13)	rmack (12)	init0 (12)	read0 (12)	init4 (6)	read1 (3)
init2	init1 (14)	iowait (12)	init0 (8)	rmack (8)	read0 (8)	write0 (8)	wmack (8)	init4 (6)	read1 (2)
init4	iowait (12)	init0 (8)	init1 (6)	init2 (6)	rmack (4)	read0 (4)	wmack (4)	write0 (4)	read1 (2)
iowait	init1 (21)	init0 (16)	rmack (15)	read0 (15)	wmack (14)	write0 (14)	init2 (12)	init4 (12)	read1 (4)
rmack	read0 (17)	iowait (15)	init1 (12)	write0 (12)	wmack (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)
wmack	write0 (17)	iowait (14)	init1 (13)	rmack (12)	read0 (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)
read0	rmack (17)	iowait (15)	init1 (12)	write0 (12)	wmack (12)	init0 (8)	init2 (8)	init4 (4)	read1 (2)
read1	init0 (4)	iowait (4)	init1 (3)	init4 (2)	init2 (2)	rmack (2)	wmack (2)	read0 (2)	write0 (2)
write0	wmack (17)	iowait (14)	init1 (13)	read0 (12)	rmack (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)

Table 2.5: Sorted edges for the states in IOFSM

From the new list state `wmack` is selected and the states `write0, init1, rmack` and `read0` have to be given close codes from it. However, in this cluster, `wmack` and `write0` are the only unassigned states. So the assignments are:

`wmack` → 0011 `write0` → 0111

Again, the head-node of the cluster, wmack, is deleted from the graph. The steps of sorting, cluster selection, and assignment are repeated till all the states are assigned codes. Clusters selected and the codes assigned in subsequent steps are given below.

`init1, init2, write0, wmack,` and `rmack`

`init2` → 0101

`rmack, read0, write0, init2,` and `init0`

no unassigned states in the cluster

`init0, init2, init4, read0` and `write0`

`init4` → 0110

`read0, write0, init2, init4` and `init1`

init0		init1 (12)	init2 (8)	init4 (8)	rmack (8)	read0 (8)	wmack (8)	write0 (8)	read1 (4)
init1		init2 (14)	write0 (13)	wmack (13)	rmack (12)	init0 (12)	read0 (12)	init4 (6)	read1 (3)
init2	init1 (14)		init0 (8)	rmack (8)	read0 (8)	write0 (8)	wmack (8)	init4 (6)	read1 (2)
init4		init0 (8)	init1 (6)	init2 (6)	rmack (4)	read0 (4)	wmack (4)	write0 (4)	read1 (2)
rmack	read0 (17)		init1 (12)	write0 (12)	wmack (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)
wmack	write0 (17)		init1 (13)	rmack (12)	read0 (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)
read0	rmack (17)		init1 (12)	write0 (12)	wmack (12)	init0 (8)	init2 (8)	init4 (4)	read1 (2)
read1	init0 (4)		init1 (3)	init4 (2)	init2 (2)	rmack (2)	wmack (2)	read0 (2)	write0 (2)
write0	wmack (17)		init1 (13)	read0 (12)	rmack (12)	init2 (8)	init0 (8)	init4 (4)	read1 (2)

Figure 2.4: Sorted edge lists after one round of assignment

no unassigned states in the cluster

`init2, write0, init4` and `read1`

`read1` → 1101

FISTEM generates the personality matrix of the PLA by replacing the present and next states in each row of the STT by their boolean codes. For the encoding obtained in this example, the optimized PLA implementation of IOFSM obtained from ESPRESSO had 16 product terms where as, on average, random assignment will result in 27 product lines. This means that

%reduction PLA area = (27 - 16) / 27 = 40%

State encoding using the Fanout-oriented algorithm also has been tried for this example and the number of product terms in the optimized two-level representation in that case is 15 (44% reduction). A large number of STG examples are encoded using these two algorithms and the results are summarized in Chapter 5. The %reduction in the cardinality of the PLA implementation, compared to the random encoding case, varied from 5 to 50%. These results clearly illustrates the efficacy of the graph construction and embedding algorithms in optimizing the two-level implementation of FSMs.

2.4 Further Minimization for PLA-Based FSMs

2.4.1 Zero-code Assignment

Both fanin and fanout algorithms discussed in the previous section are based on an average case analysis and make an assumption that the number of 1's in the next state code is Nb/2. However, this is not true in the actual case and it can take any value between 0 and Nb. In the actual implementation of the circuit, only those input combinations will be considered, which will result in a 1 in the next state or output space. In other words, an input cube which produces the zero-coded state and all-zero outputs will not map into hardware. In general, the input cubes which produce zero-coded next state, will have less constraints on minimization.

From this point of view, it can be easily concluded that assigning zero code for the state with maximum fanin can result in better optimization. In fact, this approach can be extended by assigning codes for states, in such a way that the number of 1's in the code are inversely proportional to the fanin of the corresponding state. But in the present approach, it is difficult to incorporate these constraints as it conflict with the adjacency relations derived from the fanin/fanout analysis. However, the zero code can be assigned to the state with maximum fanin without disturbing these distance relations. This can be done by shifting the embedded graph in Boolean hypercube so that the node corresponding to the maximum fanin-state will coincide with the zero code. The hamming distance between the states will not change in this process. In a stand alone version of the state assignment part of FISTEM, named FINISMA, this modification is implemented to achieve further minimization for PLA-based FSMs. The major steps involved in this process are:

1. Scan the next state space of the STG and find out the state (S_0) which occurs maximum number of times.

2. Note the Boolean code of state S_0 (which was assigned in the graph embedding step).

3. Shift the graph in Boolean hypercube so that the state S_0 is assigned null code. In this step, the new codes for the states are found by XOR-ing (bit wise) the old codes with the code of state S_0.

2.4.2 Example

The details of the STG example considered are:

Name of the machine : Lion

Source : MCNC Logic Synthesis Workshop (1987) Benchmark Set

No. of inputs : 2

No. of outputs : 1

No. of states : 4

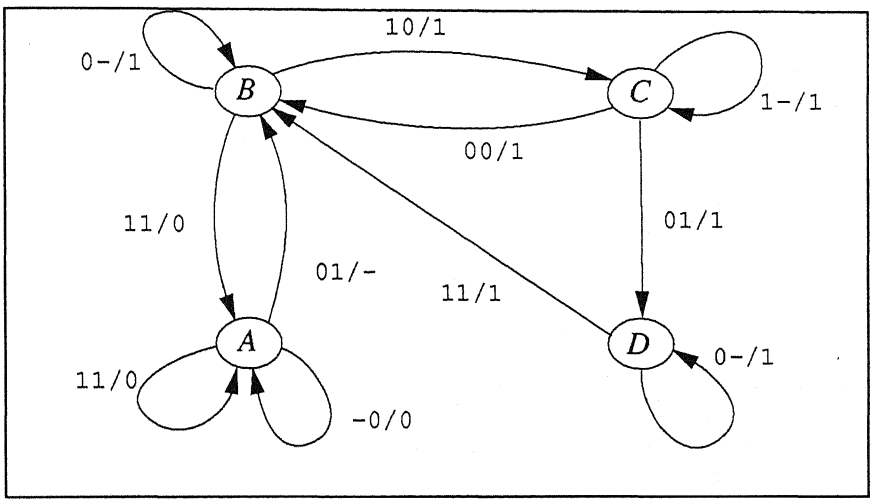

Figure 2.5: STG representation of "Lion"

First step is to assign codes to the states of FSM using the graph construction and embedding procedures. In this example, fanout-oriented algorithm is used to obtain the following assignment:

$A \rightarrow 11$ $B \rightarrow 01$ $C \rightarrow 00$ $D \rightarrow 10$

The optimized PLA implementation of "Lion", corresponding to this assignment, will have 9 product terms. In the STG representation of the FSM, the node corresponding to state B has 4 fanin edges which is the maximum in the graph. Therefore state B is selected for assigning the zero code. New set of codes are obtained by XOR-ing the old codes with the code of state $B(01)$.

$A \rightarrow 10$ $B \rightarrow 00$ $C \rightarrow 01$ $D \rightarrow 11$

The optimized PLA implementation of the FSM, for this encoding, will have only 8 product terms. So in this example, by shifting the encoded STG in Boolean hypercube so that the maxi-

mum fanin state coincides with the zero-code, an additional minimization of 11% has been achieved.

2.4.3 Analysis

This approach together with fanin and fanout oriented algorithms is implemented in FINISMA. This modification has been tried on a large number of MCNC benchmark designs and the following observations are made on the basis of that. For most of the examples considered, the zero-coding of maximum fanin-state achieved further minimization, only when tried with the fanout-oriented algorithm. In the graph embedding step of the algorithm the coding starts with zero code: The code (arbitrarily) selected for the head node of the most strongly connected cluster of the graph is zero-code. For fanin-oriented algorithm, in most of the cases, this node turns out to be the maximum fanin-state and hence, the zero-coding modification is not required.

From the results obtained for the modified fanout algorithm, it has been noted that the zero-coding step doesn't give further minimization for all STGs. For some of the examples, it even increases the number of product terms after minimization. However, for most of the large STGs considered significant reduction in number of product terms was observed. Minimization obtained for some of the examples is listed in Table 2.6. Here, the comparison is with the results obtained for fanout-oriented algorithm with no zero-coding.

Minimization indices (MI) given in the Table are parameters extracted from the STT to predict the additional minimization due to the zero-coding step. The estimation of the indices is based on the effect of various factors on this minimization. Number of fanin edges for the maximum fanin-state, number of fanin-edges for the zero-coded state in the embedded STG (before shifting the nodes), total number of edges in the STG, number of state bits, and number of primary outputs are the major factors considered in the estimation of MI. For example, for a 19 state, 7 input, and 5 output STG named keyb, the MI is calculated as follows:

State with maximum fanin	: st0
Number of fanin edges for st0(N_2)	: 112
State which is assigned zero code	: st1

Number of fanin edges for st1(N_1) : 1

Total number of transitions in the STG(N_t) : 170

Number of state bits(N_b) : 5

Number of POs(N_o) : 2

Empirical constant MI $= ((N_2 - N_1) / N_t) \times (N_b / N_o) \times 100$

$= ((112 - 1) / 170) \times (5 / 2) \times 100$

$= \underline{163}$

Example	Minimization Index	Additional minimization achieved
keyb	163	46%
tbk	59	22%
cse	31	19.5%
styr	11	14%
sse	26	9.5%
opus	0	0%
donfile	0	0%
planet	2	-2%
dk14	1	-10.5%

Table 2.6: Additional minimization achieved by zero-coding step

From Table 2.6, it is clear that the fidelity of the estimate MI is good enough to give a rough idea about the possible minimization in the encoding stage itself. If the estimated MI is less than a particular value(say 5), the zero-coding step can be skipped to eliminate a potential increase in product terms. The results obtained from FINISMA for a set of benchmark examples are summarized in Chapter 5. In most of the cases, fanout + ZC (zero-coding) approach gives better results than simple fanin/fanout oriented approach. In the present version of FINISMA, fanout + ZC approach doesn't make use of the estimate MI. By incorporating MI, the performance of this algorithm can be further improved.

Chapter 3

Easily Testable PLA-Based FSMs

In this chapter, a constrained state encoding procedure, which guarantees easy testability for PLA-based FSMs, is discussed. This approach is targeted towards the combinationally irredundant single crosspoint faults in the PLA. Internal faults in the memory elements are not considered. The two important observations, upon which this synthesis procedure is based, are explained in §3.1 and §3.2. §3.3 focuses on the constraints which are to be satisfied, in the state assignment step, to achieve easy testability. In the last section, major steps involved in the synthesis procedure are explained with an example.

3.1 Effect of crosspoint Faults on PLA Output

crosspoint fault model covers five different types of faults:

1. missing contact in the input plane (growth fault)

2. extra contact in the input plane (shrinkage fault)

3. missing contact in the output plane (disappearance fault)

4. extra contact in the output plane (appearance fault)

5. s-a-1 and s-a-0 faults at the output.

The effect of a missing contact in the input plane or an extra contact in the output plane is to enlarge the ON-set of one or more outputs of the PLA. In the first case (growth fault), the fault removes a constraint placed on the cube corresponding to the row on which the fault has occurred and thus expands the set of vertices covered by the cube. An extra contact in the output plane adds an additional cube to the output ON-set cover and thus enlarges the ON-set. On occurrence of an output s-a-1 fault, the ON-set of the corresponding output will expand so that the entire input space is covered by it. These three faults, on the basis of their effect on the ON-sets of PLA outputs, can be collectively called as GROWTH faults.

On the other hand, the occurrence of an extra contact fault in the input plane, a missing contact fault in the output plane, or a s-a-0 fault at an output has the effect of shrinking the ON-set of some of the PLA outputs. An extra contact fault in the input plane of the PLA reflects an additional constraint placed on the corresponding cube and has the effect of shrinking the set of vertices covered by the cube. A missing contact in the output plane, say at ith row and j^{th} column, reflects the removal of the cube corresponding to the ith row from the ON-set cover of the j^{th} output function. In the case of a s-a-0 fault at an output, the effect is to shrink the ON-set of that output to $\varnothing$. These three faults will be referred to as SHRINKAGE faults.

It can be noted that a single crosspoint fault which adds to (subtract from) the ON-set of some outputs, doesn't subtract from (add to) the ON-set of any of the PLA outputs. In the sequel, this observation is verified for each type of single crosspointcrosspoint faults.

In the case of a redundant fault, the effect doesn't propagate to the PLA outputs and hence the true and faulty output vectors will be same. On the contrary, if the fault is an irredundant one, at least for some input combinations the true and faulty outputs will be different. Now let us consider such an input vector i which produces true and faulty outputs o and o^F respectively for an irredundant single crosspoint fault, F, in the PLA. If the fault is a growth fault then it will add to

the ON-set of some outputs, but will not subtract from the ON-set of any output. Therefore some of the outputs with value 0 in true output vector o will have value 1 in faulty output vector o^F. But all the outputs with true value 1 will remain at 1. This means that, for GROWTH faults, the faulty output vector will dominate the true output vector ($o^F \supset o$). Let us consider an example of five output PLA for which the input combination i produces a true output vector o = [10001]. Occurrence of an irredundant GROWTH fault can change it to o^F = [11101], but never to [00000] or [01001]. Notice that both [00000] and [01001] doesn't dominate [10001].

If F is a SHRINKAGE fault then F will subtract vertices from ON-set of some of the outputs, but will not add to the ON-set of any output. Therefore some of the outputs whose value is 1 in true output vector o will have value 0 in faulty output vector o^F. Output whose value is 0 will remain at 0. This means that, for SHRINKAGE fault, the true output vector will dominate the faulty output vector ($o \supset o^F$). In the previous example, on occurrence of a SHRINKAGE fault the true output vector o = [10001] for input i can change to o^F = [00001], but never to [11111] or [01001]. Notice that only vector [00001] is dominated by the true vector [10001].

Combining the above two observations, it can be stated that *for any kind of irredundant crosspoint fault in a PLA, the true and faulty output vectors are mutually dominant.* This means that either $o^F \supset o$ or $o \supset o^F$. The same can be extended to conclude that if two vectors are mutually nondominant (in this case, the vectors are neither mutually dominant nor equal) then they can never appear as true and faulty output vectors for a single crosspoint fault in a PLA. For example pairs

o = [11100] o^F = [10000],
o = [10010] o^F = [11110], and
o = [11010] o^F = [11010]

are valid true and faulty where as pairs

o = [10101] o^F = [01010] and
o = [00001] o^F = [00010]

are not valid. It is this relationship between the true and faulty PLA outputs, upon which the current synthesis approach is based.

3.2 Relationship between State Encoding and Testability

There are three major steps in generating a test for a sequential machine. These are:

1. State justification,

2. Fault excitation, and

3. Fault propagation.

State assignment can have a profound effect on the step of fault propagation. For any irredundant crosspoint fault a present state p and input i can be found, which will propagate the effect of the fault to the PO lines or the NS lines. If it is propagated to the PO then the fault can be easily detected by justifying the state to p and applying i. On the other hand, if the effect of the fault is propagated only to NS then the machine will be taken to a faulty next state n^F instead of true next state n. In this case, the fault can be detected only if the states n and n^F are distinguishable at the PO. If the faulty state n^F also produces the same outputs as the true state n for all the input combinations, then the fault cannot be detected by applying a single input vector. A sequence of input vectors may or may not produce different output sequences for the machine. But the attempt in the present synthesis approach is to ensure *easy testability* for the FSM by making sure that a single distinguishing vector exists for any true faulty state pair. It has to be noted that even if the two states n and n^F are distinguishable in the true machine, the output can still become identical due to the fault (which has taken the FSM to state n^F instead of n). So the attempt should be to ensure different outputs for true and faulty states even in the presence of the fault.

From the discussion in the previous section, we know that two mutually nondominating vectors can never appear as a true faulty output pair for a single crosspoint fault. *If the pair of states* n *and* n^F *produce mutually nondominating POs,* o_1 *and* o_2*, for at least one primary input* i′ *in the true machine, then it is guaranteed that these state pairs will be distinguishable even in the presence of a fault.* Here, the guaranteed distinguishing vector is i'. The presence of the fault may corrupt the output o_2 and the vector produced may be $o_2^F \neq o_2$. But in a PLA, o_2^F and o_2 will be mutually dominant. Since o_1 and o_2 are mutually nondominating, o_1 has to be different from o_2^F. This means that if the true outputs for states n and n^F are mutually nondominating for input i', then it

will hold as a distinguishing vector in the faulty condition as well. The fault, in this case, could be either GROWTH or SHRINKAGE.

If the fault which has propagated to the NS lines is of type GROWTH fault then for ensuring a distinguishing vector it is not necessary to have mutually nondominating outputs for the true faulty state pair. Applying i' to the faulty state machine in state n^F may produce a corrupted primary output vector o_2^F. Since the fault is a GROWTH fault, $o_2^F \supset o_2$. *In the true machine, if* o_2, *the output corresponding to the state* n^F *and input* i', *dominate* o_1, *the output corresponding to the state n and input* i', *then it can be assured that the corrupted output* $o_2^F \neq o_1$.

Similarly, *if the fault propagated to the NS lines is of type SHRINKAGE fault, then a dominance relationship* $o_2 \subset o_1$ *for an input* i' *in the true machine will guarantee the easy testability.* Here, due to the fault the output o_2 may be corrupted to produce a vector $o_2^F \subset o_2$. The relations $o_2 \subset o_1$ and $o_2^F \subset o_2$ makes sure that the output produced by the input i' and present state n^F will never be equal to o_1.

Depending upon the type of crosspoint fault under test, the codes n and n^F will have certain relationships. Note that n and n^F subsets of true and faulty outputs of the PLA. For a GROWTH fault, the faulty state code will dominate the true state code ($n^F \supset n$). On the other hand, for a SHRINKAGE fault, true state code will dominate the faulty state code ($n \supset n^F$). These observations can be combined to state that, for a single crosspoint fault of any kind, codes n and n^F will be mutually dominant. By the same token, *a pair of states with mutually nondominating codes can never appear as faulty fault-free states in a PLA-based FSM.*

The most difficult part of the sequential test generation process is to find the distinguishing sequence. The constrained state assignment procedure used in FISTEM ensures that the faulty fault-free next state pairs are always propagated to POs within one clock cycle. The state encoding of the machine is such that all the nondistinguishable state pairs are assigned mutually nondominating codes and never allowed to appear as faulty fault-free pairs. The testability constraints which have to be imposed on the synthesis procedure to ensure easy testability are discussed in the following section.

3.3 Testability Constraints

The ultimate aim of the present synthesis approach is to ensure *easy* and full test coverage for the combinationally irredundant faults in the PLA. This require a valid justification sequence, a test vector, and a single distinguishing vector for each irredundant fault in the PLA. R-reachability of the machine to be synthesized is a most important requirement. This point is discussed in the following subsection.

3.3.1 *R*-reachability Constraint

For a nonscan design, the STG has to be R-reachable to obtain the maximum fault coverage. This means that input sequences for placing the machine in any of the 2^n possible states, beginning from the reset state, should exist. Here n is the number of encoding bits. An initial state, called reset state, is assumed to exist for every machine. The input specification of the FSM need not necessarily have $N_S = 2^x$ states, where $x = 1, 2, 3,..$etc. But excitation of faults in the PLA may require the machine to be in any of the 2^n states. To guarantee justification sequences for each of these PLA faults, all the possible 2^n states should be made reachable from the reset state. when the number of states $N_S < 2^n$, extra states have to be added to the STG. It also has to be ensured that these newly added states are reachable from the reset state to satisfy the R-reachability condition. Any of the following approach can be used for this:

- In the case of an incompletely specified FSM, the unspecified transition edges can be used to connect the added states to the specified STG.
- If the state machine is fully specified then an extra primary input may have to be added. This will double the number of minterms/cube, with half of them unspecified.
- Another way to achieve this is to split an existing state in to two or more. In this case the generated states will be equivalent to each other. This can be applied to both completely and incompletely specified machines.

Once the R-reachability condition is satisfied for an STG, justification sequences and excitation

vectors are guaranteed for all the faults in its two-level implementation.

3.3.2 Constraints on State Assignment

Constraints on state assignment for easy testability are to be extracted from the STG to be synthesized on the basis of state fan outs. This extraction process tries to identify those states which may not be distinguishable in the presence of a crosspoint fault. For each of the state pairs in the graph, one of the following constraints has to be selected depending upon the relationship between their fanouts. Let us consider a pair of states s_1 and s_2 for the following discussion.

- If the outputs produced by states s_1 and s_2 are identical for all input combinations, then the states are to be assigned mutually nondominating codes.

- If there is at least one input for which the output of s_1 dominates the output of s_2 and no input for which the output of s_2 dominates the output of s_1 then the states have to be coded in such a way that $Code(S1) \not\propto Code(S2)$. This means that either the code of s_1 should dominate that of s_2 or the two codes have to be mutually nondominant. At least for a single bit position s_1 has to dominate s_2.

- If there is at least one input for which the outputs of state s_1 and s_2 are mutually nondominant then no constraint has to be imposed on the relationship between these state codes, to guarantee a distinguishing vector.

- If there are one or more inputs for which the output of s_1 dominates the output of s_2, and there are also one or more inputs for which the output of s_2 dominates the output of s_1 then no constraint has to be considered for the state pair.

3.4 Synthesis Procedure

3.4.1 Background

The synthesis procedure proposed here is a modification of the constrained state assignment approach described in [7]. The approach in [7] was to assign mutually nondominating codes for

all state pairs which do not produce mutually nondominating POs for at least one primary input vector. Though this procedure ensures easy testability for all the irredundant crosspoint faults in the synthesized PLA based FSM, stringent nature of the constraints used makes synthesis of many practical FSMs difficult. In many cases, the approach doesn't ensure easy testability for all the testable crosspoint faults in the PLA. For example, if one state in an FSM always produces 0 at all POs, then according to the work proposed in [7] that state has to be coded in such a way that the assigned code is mutually nondominant with all $(2^n - 1)$ other state codes. This is not possible with a n bit encoding.

From the discussions of the previous two sections, it is clear that a less stringent set of constraint can be used to ensure the same level of testability. In the present approach, a pair of states has to be assigned mutually nondominating codes only when the POs of the two states are identical for all input combinations. For a state which produces zero POs for all input combinations, the present set of constraints can be satisfied by assigning zero-code. Using the set of constraints derived in the previous sections, the present synthesis procedure can cover a larger domain of practical FSM designs.

There are three major steps in generating a easily testable two-level implementation from the STG representation of the FSM. These steps are:

1. Raising the number of states in the FSM to 2^n where n is the number of encoding bits and making all the states reachable from the reset state,

2. Extraction of testability constraints for the state assignment on the basis of state fan outs and

3. Optimal state assignment meeting the dominance constraints extracted in step 2.

A more detailed discussion on the procedure is given below. A benchmark example (MCNC 87) named "dk17" is used to illustrate the flow of the algorithm.The State Transition Table representation of this FSM is shown in Table 3.1.

0	0	s1	s1	0 0 1
0	0	s2	s3	0 0 0
0	0	s3	s1	0 0 1
0	0	s4	s4	1 0 0
0	0	s5	s3	0 0 0
0	0	s6	s7	0 0 0
0	0	s7	s4	0 1 0
0	0	s8	s4	1 0 0
0	1	s3	s1	1 0 1
0	1	s7	s1	1 0 1
0	1	s1	s4	0 1 0
0	1	s2	s4	0 0 0
0	1	s5	s4	1 0 0
0	1	s4	s5	1 0 1
0	1	s8	s5	1 0 0
0	1	s6	s8	0 0 0
1	0	s1	s2	0 0 1
1	0	s3	s2	0 0 1
1	0	s2	s3	0 1 0
1	0	s5	s3	0 1 0
1	0	s6	s3	0 1 0
1	0	s8	s3	0 1 0
1	0	s4	s4	0 1 0
1	0	s7	s5	0 1 0
1	1	s3	s2	1 0 1
1	1	s7	s2	1 0 1
1	1	s5	s3	1 0 0
1	1	s6	s3	1 0 0
1	1	s8	s3	1 0 0
1	1	s1	s5	0 1 0
1	1	s4	s5	1 0 1
1	1	s2	s6	0 0 0

Table 3.1: State Transition Table Representation of example "dk17"

3.4.2 *R*-reachability

If the number of states in the FSM is close to 2^n but not quite 2^n then splitting an existing state in the machine is found to be attractive from the area penalty view point[5]. For the example "dk17" number of states is $8 = 2^3$ and all the eight states are reachable from the reset state. Since this satisfies the *R*-reachability constraint no modification of the STG is required. As an example let us consider another FSM (Figure 3.1) with three states s_1, s_2 and s_3. Note that s_1 has got fanin edges from states s_2 and s_3. To raise the number of states from 3 to 4 the edge from s_2 to s_1 can be made to go to another state s_4 whose fanout is identical to s_1. But here the problem is that the

states s_1 and s_4 are equivalent states (with identical fanout) and they have to be coded with mutually nondominating codes. This will make the state encoding step more constrained.

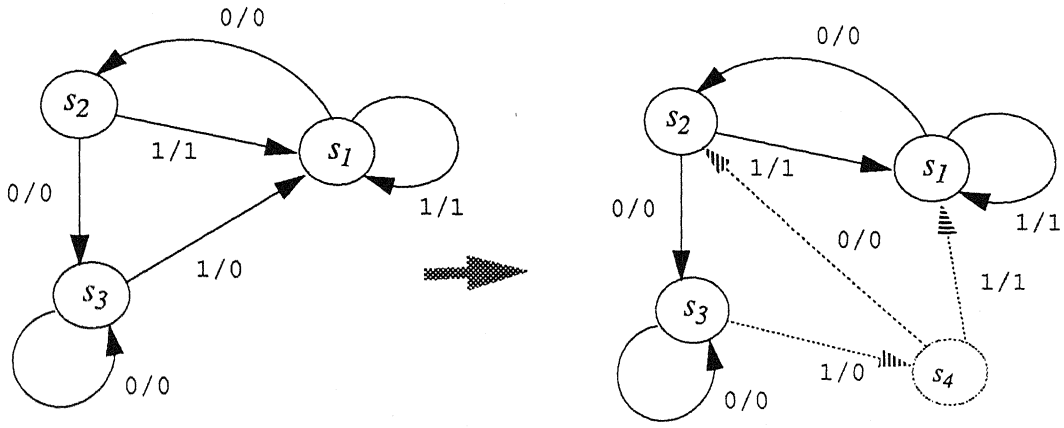

Figure 3.1: State Splitting for R-reachability

If a large number of states are to be added to the STG to raise the number of states in it to 2^n then the new states can be added to the original STG in a variety of ways. The new states can be connected in chain or separately connected from the original states. Practical FSMs designs, generally, have a large number of unspecified transitions and these can be used to attach the newly added states to the original STG. For a completely specified machine an extra input may be required.

3.4.3 Constraints Extraction

Here, the dominance relationships between each pair of states in the FSM which can guarantee easy testability have to be extracted from the STG on the basis of state fanouts. These constraints are stored in a dominance matrix with N_s rows and N_s columns and are used for guiding the subsequent graph embedding process. The N_s rows and N_s columns are associated with the N_s states. The entries in this matrix can be either 1 or 0. Except the diagonal elements, each entry in the matrix represents the a dominance relationship between a pair of states. For example an entry 1 in the i^{th} row and j^{th} column means that the code for j^{th} state should dominate the code for i^{th} state at

least in a single bit position. A zero-entry doesn't impose any constraints on the corresponding state pair. If the entries at (i, j) as well as (j, i) are 1's then the codes of the states have to dominate each other in some bit position. This means that the codes have to be mutually nondominant. The flow of the process is as follows:

- Construct the N_S x N_S dominance matrix and initialize all the entries to 1.
- For each pair of states in the machine, fanout edges are examined.
- If the intersection of input cubes/minterms for two edges, one from each state, is not null the corresponding output vectors are compared.
- Depending upon the dominance relationships between the output vectors, modify the corresponding entries in the dominance matrix. If the output of the i^{th} state dominates the output of the j^{th} state for some common input then set the entry (i, j) to value 0. If the outputs are mutually nondominant then set both entries (i, j) and (j, i) to zeroes.
- Each time after modifying the matrix entry, check the values (i, j) and (j, i). If both are zero stop comparing the fanout edges for the present pair of states.

	st1	st2	st3	st4	st5	st6	st7	st8
st1	-	0	0	0	0	0	0	0
st2	0	-	0	1	1	1	1	1
st3	0	0	-	0	0	0	0	0
st4	0	0	0	-	0	0	0	0
st5	0	0	0	1	-	0	1	1
st6	0	0	0	1	1	-	1	1
st7	0	0	0	0	0	0	-	0
st8	0	0	0	1	0	0	0	-

Table 3.2: Dominance matrix for example "dk17"

Dominance matrix for example "dk17" is shown in Table 3.2. Let us follow the procedure explained above to find the dominance relations between the codes for st2 and st4. From the STT representation of this FSM (Table 3.1), it can be seen that both the states have four fanout edges each. Each edge of st2 is compared with that of st4 to find the pair of edges, one from each state, with some input minterms common to each other. There are four such pairs of edges corresponding to the primary inputs `00, 01, 10,` and `11`. For input `10`, the POs corresponding to the edges from st2 as well as st4 are identical. This doesn't require us to change the matrix entries. For all other edge pairs output generated by st4 dominates that of st2. This implies that code of st2 should not dominate that of st4. So the entry (4, 2) is changed to 0. For states st1 and st4 the POs generated for input `00` are mutually nondominant. So both the entries for this state pair are set to zero and the rest of the edge pairs are skipped.

3.4.4 State Encoding

The state encoding approach for the easily testable FSM is similar to the optimum state assignment procedure outlined in Chapter 2, except for the additional constraints which are to be satisfied for easy testability. The fanin/fanout oriented algorithms discussed in Chapter 2 are used for the weighted graph construction. The wedge clustering algorithm for the graph embedding process is to be modified. In this step, when a cluster of states is selected, they are checked for constraints in the dominance matrix. The states in the cluster are assigned minimally distant codes satisfying the constraints. The flow of the state encoder is as follows:

- Construct the weighted graph using fanin/fanout oriented algorithms.
- Follow the wedge clustering algorithm for selection of strongly connected clusters in the STG.
- Once a group of states is selected for coding, check for the constraints in the dominance matrix and assign minimally distant set of codes to the states satisfying these constraints.

For the example "dk17", the weighted graph is constructed using the fanin oriented algorithm.

The first cluster selected from the graph, for encoding, consists of st4, st3, st5 and st1 where st4 is the head node. In the present case the head node of the first cluster cannot be directly assigned zero-code as done -*in the optimum encoding process. From the dominance matrix, it can be noticed that the code of st4 has to dominate at least four other state codes (at least in a single bit position). So the head node st4 is assigned a code 011. Rest of the nodes in the cluster are assigned minimum distant codes from the head node code, whenever possible.

st4 => 011 st3 => 001 st5 => 010, st1 =>111

In the next cluster st3, st5, st2, and st1 there is only a single uncoded state: st2

st2 => 000

The subsequent clusters and the assignments are as follows:

st2, st5, st1, and st6	st6 => 100
st5, st1, st6, and st7	st7 => 110
st1, st8, st7 and st6	st8 => 101

For this encoding, the optimized (using ESPRESSO) PLA implementation of the FSM will have 23 product terms where as the optimally encoded implementation will have 21 product terms. So the area penalty for achieving easy testability, in the present case, is 9.5%.

GENSEQ, the test generation program of FISTEM, was used to obtain the test sequences for the present example. 63 test sequences of average length 3.2 were generated with a total fault coverage of 96.4%. Maximum length of the sequence was 5. Using the PLA test pattern generation program of FISTEM (PLATEST), tests were also generated for the combinational part of the synthesized FSM. 31 test patterns were produced to obtain the maximum possible fault coverage of 96.4%. So the fault coverage obtained for the synthesized nonscan FSM is same as that of a full scan design with the same combinational part. At the current point of time, graph embedding step of the testability synthesis tool is not fully automated and needs manual intervention.

Chapter 4

Test Generation

This chapter addresses the problem of test generation for easily testable PLA-based FSMs. A procedure for generating test sequences to detect all the irredundant crosspoint faults in PLA is described. The test sequences are obtained by concatenating a fault-free justification sequence, the input excitation vector, and a distinguishing vector (which is guaranteed to exist for an easily testable FSM). GENSEQ, the test sequence generator of FISTEM, implements this procedure to obtain maximum fault coverage for the PLA-faults. Internal faults of the memory elements are not considered.

An exact algorithm for test pattern generation and simple breadth first search on the STG are used to generate the sequences. A new version of PLATEST, a PLA test pattern generator has been implemented to generate a compact set of test vectors which covers all the testable cross-point faults in the PLA. A brief description of this pattern generation strategy is given §4.1. The high level structure of GENSEQ and the sequence generation approach are explained in §4.2. In §4.3 optimized two-level implementation of a small FSM is used for illustrating the flow of the test generation process

4.1 Test Pattern Generation

In this section, a brief description of the PLA test pattern generation approach used in the present work is presented. This approach is targeted towards the crosspoint fault model of programmable arrays. The algorithm described is based on two basic principles: Path sensitization and fault simulation.

The work described in [19] includes implementation of these algorithms in C in PC/DOS environment. This program, named PLATEST, generates a set of test vectors to cover most of the PLA crosspoint faults, from an optimized PLA representation generated by ESPRESSO. In the present work, a new version of PLATEST has been implemented as a component of FISTEM environment. In addition to the deterministic algorithms implemented in [19], some heuristics in the area of don't-care bit fixing, Backend Fault Simulation, and the fault processing order also have been incorporated. The current version generates a compact set of test vectors to cover maximum crosspoint faults in the PLA. The algorithm is exact, *i.e.*, every testable crosspoint faults is tested.

If the input PLA representation is a prime and irredundant one, then the tests generated by PLATEST will detect all the Growth and Disappearance faults in the PLA. Logic minimization programs like ESPRESSO can ensure prime and irredundant covers, and hence 100 percent testability for the stuck-at faults in the PLA. However, since the crosspoint fault model is a super set of stuck-at fault model, PLAs implementing prime and irredundant covers may not be testable for all crosspoint faults. However a large percentage of these faults can be made testable via optimization [26]. Only a few Shrinkage and Appearance faults will be left undetected.

The strategy used, in PLATEST, for test generation consists of three major steps:

1. Identify all the crosspoint faults in the PLA from its personality matrix representation and prepare a fault list.

2. For each fault in the list, find a test vector using the deterministic algorithms.

3. Fault simulate the test pattern generated in the step2 and remove all the faults covered by the pattern from the fault list, before the next fault is considered in step2 for pattern generation.

A brief discussion on each of these steps are and the various heuristics used, are presented in the following subsection. A detailed discussion of the algorithms can also be found in [19] and [26].

4.1.1 Fault List Preparation

From the input personality matrix representation of the PLA, a list of all Growth, Shrinkage, Appearance and Disappearance faults is prepared. Output stuck-at faults are not considered in the present implementation. However, most of these faults are automatically covered by the tests for other types of faults. For an input PLA implementation shown in Table 4.1, the fault list generated by PLATEST is given in Table 4.2. This example is a PLA implementation of NSL and OL for a small FSM[7] whose states are encoded using the testability-driven synthesis approach, discussed in the previous chapter.

	0 1 2 3	0 1 2 3 4 5
0	0 0 1 -	1 0 0 0 0
1	- 1 0 0	0 1 1 0 0
2	1 - 1 1	0 0 0 0 1
3	1 - 0 1	1 0 0 1 0
4	- - 0 1	0 1 0 0 0
5	- - 1 0	1 1 0 1 0
	AND PLANE	OR PLANE

Table 4.1: Personality Matrix Representation of the PLA for the FSM example[7]

For each contact in the input cubes, a Growth fault is considered where as each don't-care bit in the input plane maps into a pair of Shrinkage faults: One for the extra contact in the 1s line and the

other for the 0s line. For each 1s in the output plane a Disappearance fault is included. Similarly for 0s in the output plane, Appearance faults.

Plines	INPUTS				OUTPUTS				
	0	1	2	3	0	1	2	3	4
0	G	G	G	S	D	A	A	A	A
1	S	G	G	G	A	D	D	A	A
2	G	S	G	G	A	A	A	A	D
3	G	S	G	G	D	A	A	D	A
4	S	S	G	G	A	D	A	A	A
5	S	S	G	G	D	D	A	D	A

Table 4.2: List of Cross Point Faults for the PLA implementation of FSM example[7]

4.1.2 Deterministic Algorithms

The algorithms used in PLATEST for test generation are deterministic in nature and are based on the principle of path sensitization. Here the approach is to obtain at least a minterm which on applying to the PLA excites the fault under consideration and propagates the effect of the fault to the output. Even though the steps of fault excitation, error propagation and line justification are not explicit, the procedure discussed below does the same to obtain test patterns. The program uses a complex fundamental operation called sharp (#), very extensively, to find the set difference between various Boolean covers.

a) Growth Faults

On occurrence of a Growth fault in i^{th} row of PLA, the cube c_i corresponding to the that row will change to $c_i^F \supset c_i$. The cube $c_i^F - c_i$ covers the vertices added to c_i by the fault. In path sensitization approach the first step is to apply an input which will generate opposite logical values at the fault site, for true and faulty conditions. The inputs of the PLA has to be set to a value corresponding to any of the vertices in $c_i^F - c_i$ to excite the fault. For the propagation of this effect of

the fault to an output, all other product terms asserting that output should be 0s. So for justifying the line of propagation of a fault to an output, say k^{th}, the test should not be covered by the *ON* set of that output. Hence, the set of vertices added to the *ON*-set of k^{th} output (which will be the set of tests for the fault under consideration) can be found from the set difference $(c_i^F - c_i)$ # $RON_k(c_i)$. Here, $RON_k(c_i)$ is the set of all the cubes which defines the ON-set of k^{th} output except the i^{th} cube c_i. If the inputs of the PLA are set to a value corresponding to any of these vertices, a faulty PLA will have a 1 in the k^{th} output while the good PLA has a 0 in it.

The set difference between c_i^F and c_i consists of a single cube and can be easily found from the cube c_i. On occurrence of a Growth fault at position (i, j) will change the cube c_i in its j^{th} input, from a care input value to don't care. This will double the number of vertices covered by the cube. The set of vertices added, $c_i^F - c_i$, can be easily found by replacing the j^{th} input of c_i to its logically opposite value. As an example let us consider the possible Growth fault at position (0, 0) in the example PLA given in Figure 4.1. The cube corresponding to $i = 0$, c_0 = `[001-]` will change to c_0^F = `[-01-]`, due to the occurrence of this fault. The set of vertices added to c_0 is c_0^F - c_0 = `[101-]`. To obtain a test vector for this fault a nonempty value for $(c_0^F$ - $c_0)$ # $RON_k(c_{0i})$ should exist for some output k. In the present case, for output ($k = 0$), the complete *ON*-set is (from Table 4.1)

ON_0		= `[[001-], [1-01], [--10]]`
Reduced-*ON*-by-c_0	$RON_0(c_0)$	= `[[1-01], [--10]]`
So the test set for the G(0, 0) is		= `[101-] # [[1-01], [--10]]`
		= `([101-] # [1-01]) # [--10]`
		= `[101-] # [--10]`
		= `[1010]`

The single minterm test set `[1010]` can detect the Growth fault at (0, 0) and is included in the test vectors (Table 4.4)generated by PLATEST for the example PLA.

b) Shrinkage Fault

The set of tests which detect a Shrinkage fault which causes a cube c_i to become $c_i^F \subset c_i$, is given by the set of vertices of the cubes in $(c_i - c_i^F)$ # $RON_k(c_i)$, where k is the output where the fault can be detected. Here the cube $(c_i - c_i^F)$ covers the vertices which are removed from cube

c_i due to the occurrence of the fault. The sharp operation identifies the vertices which are covered only by $(c_i - c_i^F)$. For the PLA inputs corresponding to these vertices, the k^{th} output of the PLA will have 1 and 0 outputs for true and faulty conditions respectively.

Considering a Shrinkage fault at (0, 3 : 0) in the example PLA of Table 4.1, it can be seen that the occurrence the of this fault shrinks the input cube c_0 = [001-] to c_0^F = [0010]. Number of vertices covered by the cube has reduced from 2 to 1. The difference $(c_0 - c_0^F)$ can be easily found by replacing the $(j = 3)^{rd}$ bit of c_0^F to its logically opposite value.

i.e., $(c_0 - c_0^F)$ = [0011]

Reduced-*ON*-by-c_0 $RON_0(c_0)$ = [[1-01], [--10]]

The set of tests for fault (0, 3 : 0) = [0011] # [[1-01], [--10]]

= ([0011] # [1-01]) # [--10]

= [0011] # [--10]

= [0011]

In this case also a single minterm test is obtained and it will detect the fault at the 0^{th} output. In the case of a Shrinkage fault it is possible that the test set derived for all the output columns are $\varnothing$. In such cases PLATEST includes those faults into a list of undetectable faults.

c) Appearance and Disappearance Faults

A crosspoint fault at location (*i*, *k*) of the output plane of the PLA can add or remove the i^{th} cube c_i in the *ON*-set of k^{th} outputs. If the fault is Disappearance fault then the vertices covered by the cube c_i will be removed from the *ON*-set of the output where as in the case of Appearance fault vertices will be added. Any *free* minterm covered by the cube c_i can detect the occurrence of any of these faults. The free minterms in the cube c_i corresponds to those vertices which are realized only by the i^{th} product line of the PLA. For a Disappearance fault at (*i*, *k*) the set of tests can be found as c_i # $RON_k(c_i)$. For Appearance fault, it will be c_i # ON_k. As examples let us consider the faults at (5, 1) and (5, 2) in the output plane of the example PLA. For the Disappearance fault at (5,1) c_i = c_5 = [--10].

For the output ($k = 1$)

$RON_k(c_i) = RON_1(c_5)$ = [[--01], [-100]]

Set of test vectors = [--10] # [[--01], [-100]]

= ([--10] # [--01]) # [-100]

= [--10] # [-100]

= [--10]

Any of these four patterns can be used to detect the Disappearance fault at (5, 1). Similarly for the Appearance fault at (5, 2) the test set is

$c_i \# ON_k(c_i)$ $= c_5 \# ON_2(c_5)$

= [--10] # [-100]

= [--10]

The fault can be detected at the second output of the PLA by applying any of these four vectors. A true PLA response will be 0 while the faulty output is 1. It should be noted that the Boolean optimization step using ESPRESSO doesn't guarantee a free minterm for all the cubes in the optimized PLA. However, ESPRESSO ensures that every cube connected to an output will have at least one free minterm w.r.t. that output. But still some of these cubes can be covered entirely by a set of cubes connected to other outputs in the PLA. As a result when some of the Appearance faults are undetectable full testability is guaranteed for the Disappearance faults.

4.1.3 Fault Simulation

The test generation could be quite expensive and the number of tests large if the deterministic process is used for all the faults under consideration. PLATEST uses fault simulation to discover all the faults which are covered by a test found by deterministic algorithm for a particular fault. This has the effect of reducing the number of faults to be considered by the deterministic algorithm.

The fault simulator implemented in FISTEM uses the following approach for finding all the faults covered by a test vector.

- First, the don't care bits in the pattern are fixed and the test is considered as a single minterm input to the PLA.
- Each of the input cubes in the PLA are examined to find all the product lines which are asserted by the test pattern under consideration.
- For each output note the logical values taken by the product lines which are connected to it.
- If all the inputs (product lines) to the OR gate corresponding to an output is zero, then some of the Appearance faults in that output column and Growth faults in its fanin product lines may be detectable.
    - If all the product lines connected to the k^{th} output takes 0 value and a row (say i^{th}), which covers the vertex corresponding to the input pattern, doesn't have a contact at (i, k) in the true machine then the fault A(i, k) will be covered by the test vector under consideration.
    - If all the product lines connected to the k^{th} output are evaluated to 0 and a cube c_i corresponding to one of these p-lines is *distance-1* from the input pattern then the Growth fault at the conflicting input line (where a 0/1 clash occur) of c_i will be covered.
- If all the fanin cubes for an output takes value 0, except for a single cube corresponding to (say i^{th})a product line, then
    - The Disappearance fault at (i, k) will be detected by the test vector under consideration.
    - If there are don't care bits in the cube c_i, then one of the two possible Shrinkage faults at those crosspoints will be covered by the pattern under consideration.If the j^{th} input of the cube c_i is "-" and that of the test pattern is 0(1), then Shrinkage faults at (i, j : 1) (at (i, j : 0)) will be covered by the test.

Tests generated by PLATEST for the example PLA is given in Table 4.4. Out of the total of 62

faults in the PLA, only 14 were considered by the deterministic algorithms. Rest of the faults were covered by these deterministically found patterns. This clearly shows the efficacy of the fault simulation approach in generating a compact set of test vectors, simultaneously reducing the run time.

Crosspoint Faults covered by the test vector [1010]			
Growth	Shrinkage	Appearance	Disappearance
(2, 3)	(5, 0 : 0), (5, 0 : 1)	(5, 2), (5, 4)	(5, 0), (5, 1), (5, 3)

Table 4.3: The List of Crosspoint faults covered by test vector [1010]

4.1.4 Heuristics

In the current version of PLATEST, some powerful heuristics in the area of fault processing order, Backend Fault Simulation (BFS), and don't-care bit fixing are incorporated for improving the run-time performance of the program and the test set minimality. A discussion on each of these is given below:

a) Don't Care Bit Fixing

Before performing fault simulation, all don't care bits in a test pattern have to be fully specified. The fault coverage of the test vector and the subsequent compaction of the patterns are dependent on the step of fixing the don't care bits. For the vectors generated for G and A faults using deterministic approach, the "-" bits are replaced by randomly selected 0 and 1 bits. In the case of S and D tests the don't care bits are set to address the untested S faults in the same cube. By doing so, the first S or D test considered for a cube will address at least half of the S tests on it. If both the Shrinkage faults at a crosspoint in that cube are not tested the present vector is split into two so that the two tests will address at least half of the S tests on it. If both the S faults at the don't care bit position of a test pattern is already tested then the bits are randomly fixed.

b) Backend Fault Simulation (BFS)

Though a test is not redundant when it is generated, at the end of test generation some tests may become redundant. Finding the essential tests from the generated set is a minimum covering prob-

lem and is *NP*-hard. A simple heuristic [24] called Bottom-Up Backend Fault Simulation is used to reduce the size of the test set generated by PLATEST. Here, tests are fault simulated in the reverse order of their generation. Faults covered by a test vector are removed from the list. If a vector is found not to cover any remaining fault it is removed from the set. For the examples tried on PLATEST an average test set size reduction of around 10 percent was obtained. In the present example number of patterns in the test set is reduced from its original number 14 to 13 via BFS. For some examples, a further reduction in the test set size was observed by once again performing BFS on the test set. However the minimization achieved in the second round was very small and the run time was comparable with that of first run.

c) Fault Processing Order

It has been noted that the A and D faults in a PLA have got larger test sets compared to the G and S fault tests. This can be explained in terms of the closeness of these fault sites to the outputs. In general, for any circuit, the faults closer to the output will create larger impact on the functionality of the circuit. So the probability of a random pattern covering a fault is more in the case of A and D faults. From this point of view, the fault processing operations are ordered in such a way that G and S faults are considered first and then the A and D faults. In the example considered in this section, the first 14 vectors obtained from the deterministic algorithms covered all other testable faults in the PLA. It is worth mentioning that A and D faults in this example are never considered by the deterministic algorithm. All these output plane faults are covered by the tests generated for G and S faults.

Test vectors generated by PLATEST for the PLA example is given in Table 4.4. The ratio of the "used crosspoints" in the PLA to the number of patterns generated for the example is 26/13 = 2, which is an index of the efficacy of the program in generating a small set of test to cover all the testable crosspoint faults in the PLA. There are only two undetectable faults in the present example: A(3, 1) and S(0, 3 : 1) and rest of the 62 faults are covered by 13 test patterns. Hence the fault coverage achieved by PLATEST for this example is 96.8 percent. For Growth and Disappearance faults full coverage of the fault is obtained.

Sl. No.	Test Pattern	True Output
1	1010	11010
2	0110	11010
3	1001	11010
4	1111	00001
5	1100	01100
6	0100	01100
7	0011	10000
8	1000	00000
9	0101	01000
10	1101	11010
11	0000	00000
12	0111	00000
13	1011	00001

Table 4.4: Test Patterns and True Outputs for the PLA implementation of FSM example[7]

4.2 Test Sequence Generation

The sequential test generation program of FISTEM, named GENSEQ, accepts the encoded State Transition Table representation of easily testable FSMs and generates a compact set of test sequences to achieve maximum fault coverage. Only easily testable FSMs are targeted in the current implementation and it is assumed that at least one distinguishing vector exists for each true-faulty state pair in the input FSM. It is also assumed that the cardinality of the input graph is $2x$ where $x = 1, 2, 3...$etc. and all the states are reachable from an initial state (reset state).

4.2.1 Easy testability

In GENSEQ, sequences are obtained by concatenating the justification sequence and the input portion of the test pattern generated by PLATEST. If the effect of the fault doesn't propagate to the PO then a single distinguishing vector for the true-faulty state pair also will be appended to it.

A conventional sequential test generator also does the same to obtain test sequences for optimized state machines. But in that case a sequence of input vectors may be required to distinguish between true and faulty states. Such a distinguishing sequence may not exist if the true and faulty states are equivalent. Since no *a priori* knowledge of the length of the distinguishing sequence is available, a large amount of effort may be wasted in searching among all the possible sequences of different lengths. When the true and faulty states are equivalent, the program has to search quite exhaustively before giving up. Another possibility is that even if such a distinguishing sequence exist in the true machine, it may not hold under the fault condition.

In the present case, at least one distinguishing vector is guaranteed to exist for each true faulty state pair. The considerably reduces the search time. In conventional test sequence generation, the entire sequence generated has to be fault simulated to make sure that the fault under consideration doesn't corrupt the sequence. The constrained synthesis procedure also ensures that the guaranteed distinguishing vectors always hold under fault conditions. therefore only justifications are to be fault simulated in the present problem.

4.2.2 An Overview of the FISTEM Test Generator

An overview of the test sequence generation part of FISTEM is shown in Figure 4.1. The input to the system is the encoded STT representation of an FSM synthesized using the testability driven approach discussed in the previous chapter. GENSEQ uses the Boolean optimization program ESPRESSO for obtaining a minimized, prime and irredundant, two-level implementation of the input FSM. All crosspoint faults of type growth, disappearance, and output stuck are guaranteed to be testable via this step. PLATEST is then used to generated a compact set of test vectors for the optimized PLA implementation. PLATEST also provides the list of crosspoint faults covered by each test pattern. For each of these faults GENSEQ generates a valid test sequence using

the corresponding test pattern and the input FSM description.

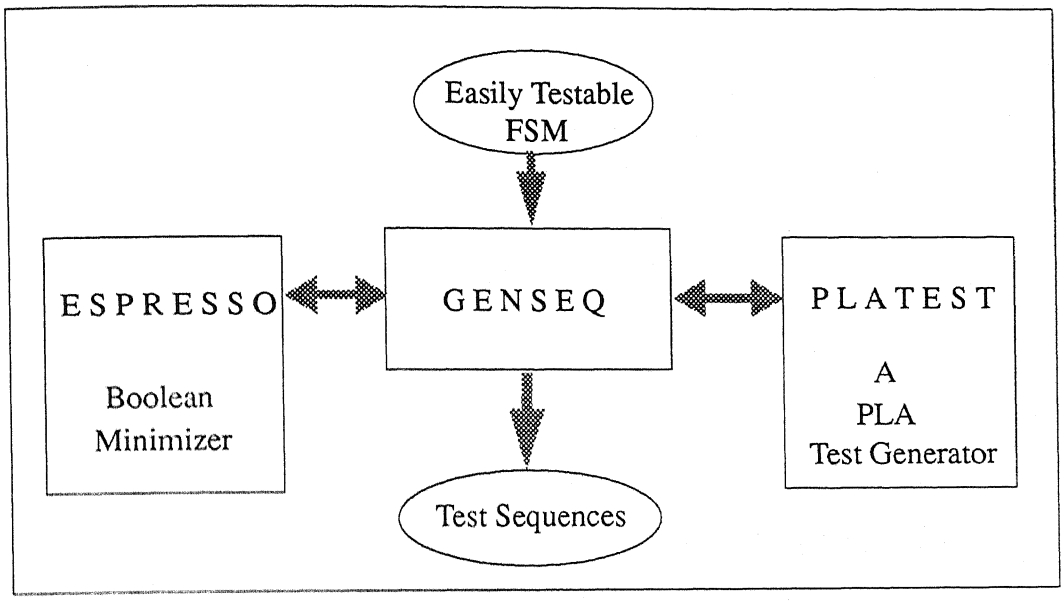

Figure 4.1: An overview of the FISTEM test generator

4.2.3 Test Generation Strategy

A brief algorithmic description of GENSEQ's test generation procedure is given below.

1. Obtain the minimized two-level implementation of the input FSM using ESPRESSO. The input file containing the encoded STT representation of the easily testable FSM in "kiss" format is to be directly passed to the optimizer.

2. Run PLATEST to obtain a set of test vectors for the optimized PLA implementation generated by ESPRESSO. Lists of crosspoint faults covered by each test pattern are also to be generated.

3. Generate a valid test sequence for each of the irredundant crosspoint faults using the following technique.

   - Find the test pattern which covers the fault under consideration. The first N_i bits of this pattern gives the excitation vector. Rest of the pattern is the code of the state to

which the machine has to be justified before applying the excitation vector.

- Do a simple breadth first search on the input STG to obtain a potential justification sequence.
- Append the excitation vector to the potential justification sequence.
- Fault simulate the sequence. If it is invalid in the presence of the fault under consideration then truncate the sequence after the first corrupted edge in it. If the sequence is valid only the last edge in the sequence will be corrupted.
- The last vector in the resulting sequence excites the fault under consideration.If the sequence is a test sequence by itself add it to the final list of sequences (if it is not already in the list). In this case the effect of the fault goes to the POs.
- If the effect of the fault is not propagated to the PO lines and a true faulty state pair is generated then find a distinguishing vector for the state pair. While implementing this step the following points are to be kept in mind.
    - In the approach described in [7], distinguishing vectors always had to be an input for which the true and faulty states generates mutually nondominating outputs.
    - For any single crosspoint fault the true and faulty state codes will be mutually dominating.
    - In the present procedure, the approach is to search on the input STG to find an input pattern for which the dominance relationship between the two state codes and that between the corresponding outputs are similar.
    - Input vector for which the state pair under consideration asserts mutually non-dominating outputs is also a valid distinguishing vector.
- Append the distinguishing vector to the simulated (fault free) sequence. If it is a new sequence then add it to the list of sequences.

4.3 Example

Let us consider a small FSM example[7] encoded using the testability-driven synthesis discussed in Chapter 3. Each step of the *easy* test generation process of GENSEQ can be illustrated

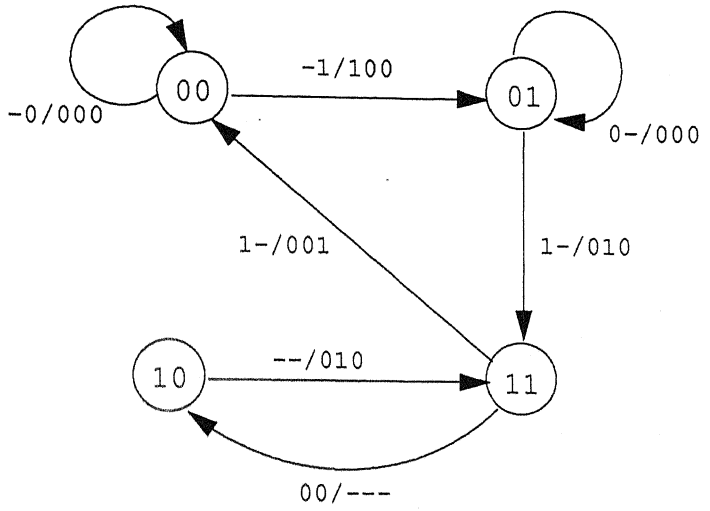

Figure 4.2: Encoded STG representation of the FSM example[7]

as follows.The encoded STG for this 4 state FSM example is shown in Figure 4.2. A two-level prime and irredundant implementation of this two input, three output FSM is obtained using ESPRESSO (Table 4.1). Note that this is the same PLA which has been considered in §4.1. The rows and columns of the PLA are numbered so that different crosspoint faults can be easily referred to. Test patterns generated by PLATEST for this implementation is given in Table 4.4. It is a compact set of test vectors with the ratio of number of "used cross points" to the number of vectors equal to 2. Lists of faults covered by each pattern are also obtained from the test generator.

Now let us consider a growth fault at (2, 3) and generate a test sequence for it. From the lists generated by PLATEST, the test vector which covers this fault is found to be [1010]. Only the first two bits of the PLA inputs are accessible. Last two bits shows the state to which the machine has to be justified before applying the excitation vector [10]. A breadth first search on the encoded input STG starting from the reset state 00 will give the potential justification sequence for state 10 as [-1], [1-], [00]. This, in fact, will take the machine from reset state to state [10] through [01] and [11] as shown in Figure 4.3. Appending the excitation vector [10] with it,

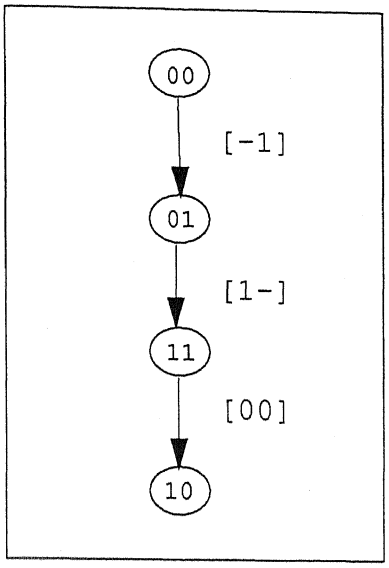

Figure 4.3: Search path for finding the justification sequence of state 10

we obtain a potential test sequence [-1], [1-], [00], [10] which has to be fault simulated. Simulation result for this sequence will show that the first corrupted edge in the sequence, for the growth fault at (2, 3), is the excitation fault itself. PLA input corresponding to this edge is [1010]. The fault under consideration changes the output of the PLA from [11010] to [11011]. It can be noticed that the primary output of the FSM also has changed from [010] to [011]. Hence a distinguishing vector is not necessary in this case. So the sequence [-1], [1-], [00], [10] is a valid test sequence to test the growth fault at (2, 3) and is included in the list of test sequences (entry 17 in Table 4.5).

Considering another fault, of disappearance type, covered by the test vector [1001] the results obtained in each intermediate step are given below:

Fault location	: (3, 0) in the output plane
Excitation vector	: [10]
State to which the FSM is to be justified	: [01]
Justification sequence	: [-1]
Sequence for fault simulation	: [-1], [10]

In this case also, the first corrupted edge in the sequence is the one corresponding to the excita

Sl. No.	Test Sequences	True Output
1	[11]	100
2	[01]	100
3	[10]	000
4	[-1], [10]	010
5	[11], [1-]	010
6	[11], [11]	010
7	[10], [11]	100
8	[-1], [01]	000
9	[-1], [11]	010
10	[00], [-1]	100
11	[-1], [10], [1-]	001
12	[-1], [1-], [11]	001
13	[-1], [1-], [00]	000
14	[-1], [01], [1-]	010
15	[-1], [01], [11]	010
16	[-1], [1-], [10]	001
17	[-1], [1-], [00], [10]	010
18	[-1], [1-], [00], [01]	010
19	[-1], [1-], [11], [11]	100
20	[-1], [1-], [11], [-1]	100
21	[-1], [1-], [00], [1-]	010
22	[-1], [1-], [00], [-1]	100
23	[-1], [1-], [01], [-1]	100
24	[-1], [1-], [10], [-1]	100
25	[-1], [1-], [00], [10], [1-]	001

Table 4.5: Test Sequences and True Outputs for the FSM example[7]

tion vector [10]. The true and faulty PLA outputs corresponding to this input [1001] are [11010] and [01010]. Here the effect of the fault doesn't appear at the PO as the last $N_o = 3$ bits are same in both the output vectors. Instead, a faulty state [01] which is different from the true state [11] has been generated. Now a distinguishing vector to detect the fault at the PO in the next clock cycle is to be found. As discussed in the previous section an input vector for which the output of state [11] dominates the output of state [01], at least in a single bit position, has to be found. Examining the fanout edges of [01] and [11] in the STG (Figure 4.2), it can be found that [1-] is the only input cube for which the outputs of the two states are assured to be different even in the true machine. For this input outputs corresponding to the true and faulty states are [001] and [010]. Note that output for state [11] dominates that for state [01] in the third bit position of the output vector. Hence a valid distinguishing vector [1-] exists for the disappearance fault under consideration. Appending this to the earlier sequence we obtain a test [-1], [10], [1-] (entry no. 11 in Table 4.5), as guaranteed by the testability-driven synthesis procedure.

Chapter 5

Results and Conclusions

In this thesis the problems of synthesis and testing of easily testable PLA-based Finite State Machines have been addressed. A new system, named FISTEM, which support testability synthesis and test generation of PLA-based FSMs has been proposed. The system also supports optimum state encoding and PLA test pattern generation.

A graph construction and embedding algorithm has been implemented for optimum state encoding of FSMs. This approach was originally proposed for multilevel logic implementation. In the present work, this algorithm has been used for encoding a large number of PLA-based state machines. The results obtained (Table 5.1) clearly demonstrates the efficacy of this technique in minimizing the number of product terms in an optimized two-level implementation.

Some modifications on these algorithms to achieve further minimization have also been tried. A zero coding modification (ZC) on fanout oriented algorithm has improved its performance substantially. Additional minimization up to 40-50% has been achieved for some of the benchmark

examples considered. From Table 5.1, it can be seen that for most of the examples Fanout + ZC approach obtains better optimization. However, at least for a few cases, the ZC modification even increases the number of product terms in the optimized PLA. A concept of Minimization Index (MI) has been proposed to tackle this problem. Here the attempt is to predict the possible

example	Number of Product Terms				% reduction in Product Terms[a]		
	Random[b]	Fanin	Fanout	Fanout + ZC	Fanin	Fanout	Fanout + ZC
bus arbtr.	32	29	31	29	9.4%	3.1%	9.4%
bbara	30	26	28	28	13.3%	6.7%	6.7%
cse	63	49	56	45	22.2%	11.1%	28.6%
dk14	40	30	28	31	25%	30%	22.5%
dk15	22	21	18	18	4.5%	18.2%	18.2%
dk16	91	83	74	74	8.8%	18.7%	18.7%
dk17	23	21	26	24	8.7%	-13.1%	-4.3%
dk27	11	10	9	9	9.1%	18.2%	18.2%
donfile	68	57	54	54	16.2%	20.6%	20.6%
ex1	64	55	56	56	14.1%	12.5%	12.5%
fetch	-	35	33	33	-	-	-
opus	27	16	15	15	40.7%	44.4%	44.4%
keyb	110	58	101	55	47.3%	8.2%	50%
lion	10	8	9	8	20%	10%	20%
planet	103	97	98	100	5.8%	4.9%	2.9%
sand	107	102	95	86	4.7%	11.2%	19.6%
sse	35	33	32	29	5.7%	8.6%	17.1%
styr	130	106	123	106	18.5%	5.4%	18.5%
tbk	204	185	228	178	9.3%	-11.8%	12.7%

a. Compared to the Random encoding case
b. An average value from a large number of random encodings

Table 5.1: Optimum State Assignment Results

minimization from the ZC modification, in the encoding stage itself. This can help the encoder to skip the ZC step for those examples for which the estimated MI values are poor. The present version of FINISMA, a state encoder implementing the Fanout + ZC algorithm doesn't make use of the estimate MI. By incorporating it the performance of the program can be further improved.

The work presented in [7] has proposed a constrained state assignment approach for ensuring easy testability for PLA-based FSMs. But the stringent nature of the testability constraints used makes the state encoding of the machine difficult, and in some cases impossible. In the present work a less stringent set of constraints are proposed for achieving the same goal.

The effect of single crosspoint faults in PLAs on their input-output relations and, on the basis of that, the relationship between state encoding and testability have been studied. The necessary constraints required for testability synthesis of PLA-based FSMs have been formulated. FISTEM makes use of this synthesis procedure and the Boolean minimization program ESPRESSO to realize easily testable FSMs. At the current point of time graph embedding step of the testability synthesis tool is not fully automated and needs manual intervention.

Test sequence generator for the easily testable PLA-based FSM has been implemented using PLA test pattern generation and simple breadth first search in the STG. An exact algorithm which assures maximum fault coverage with a compact set of test patterns has been implemented. Some powerful heuristics are also used in the pattern generation program PLATEST to ensure reduced run time and test set minimality. The results obtained for Pattern and Sequence generation tools of FISTEM are given in Table 5.2 and 5.3. From the tables it can be seen that both programs tests all the testable crosspoint faults. The trend indicated by the column for area overhead in Table 5.3 is that for larger FSM s the area penalty for improved testability will be small.

example	#used cross points N_u	#fault s	#test vectors N_v	#undetectable faults S + A = tot	Fault coverage in %	The ratio N_u/N_v
dev	26	62	13	1 + 1 = 2	96.77	2
mc	34	107	17	0 + 4 = 4	96.26	2
dk17	132	277	31	5 + 5 = 10	95.59	4.258

Table 5.2: Test Pattern Generation Results

example	#faults	#test seq	seq. length avg.	seq. len max	Fault covrge %	Area overhead
dev	62	25	2.9	5	96.77	20%
mc	107	23	2.5	4	96.26	15%
dk17	277	63	3.2	5	95.59	10%

Table 5.3: Test Sequence Generation Results

References

[1] V. D. Agarwal and K. T. Cheng. Threshold-value simulation and test generation. In F. Lombardi and M. Sami, editors, *Proc. NATO ASI on Testing and Diagnosis of VLSI and ULSI*, pages 311-323, Como, Italy, 1987. Kluwer Academic Publishers.

[2] K. Bartlett, R. K. Brayton, G. D. Hachtel, C. R. Morrison, R. M. Jacoby, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. Multilevel logic minimization using implicit don't cares. *IEEE Transactions on Computer-Aided Design*, 7(6):723-740, 1988.

[3] R. K. Brayton, G. D. Hachtel, Curt MucMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, MA, 1984.

[4] M. A. Breuer and A. D. Friedman. *Diagnosis & Reliable Design of Digital Systems*. Computer Science Press, Rockville, MD, 1976.

[5] S. Devadas, Private communication, June 1991.

[6] S. Devadas and K. Kuetzer. A unified approach to the synthesis of fully testable sequential machines. *IEEE Transactions on Computer-Aided Design*, 10:39-50, 1991.

[7] S. Devadas and H.-K. Tony Ma. Easily testable PLA-based finite state machines. *IEEE Transactions on Computer-Aided Design*, 9:604-611, 1990.

[8] S. Devadas, H.-K. Tony Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: State assignment of finite state machines targeting multilevel logic implementations. *IEEE Transactions on Computer-Aided Design*, 7(12):1290-1300, 1988.

[9] S. Devadas, H.-K. Tony Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. A synthesis and optimization procedure for fully and easily testable sequential machines. *IEEE Transactions on Computer-Aide Design*, 8:1100-1107, 1989.

[10] S. Devadas, H.-K. Tony Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Irredundant sequential machines via optimal logic synthesis. *IEEE Transactions on Computer-Aided Design*, 9:8-18, 1990.

[11] D. L. Dietmeyer. *Logic Design of Digital Systems*, Chapter 5, pages 363-491. Allyn and Inc., third edition, 1988.

[12] E. B. Eichelberger and E. Lindbloom. A heuristic test-pattern generator for programmable logic arrays. *IBM Journal of Research and Development*, 24(1):15-22, 1980.

[13] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[14] S. J. Hong and D. L. Ostapko. Fault analysis and test generation for programmable logic arrays (PLA's). *IEEE Transactions on Computers*, 28:617-626, 1979.

[15] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, 1978.

[16] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment of finite state machines. *IEEE Transactions on Computer-Aided Design*, CAD-4(7):269-285, 1985.

[17] B. Prakash and M. M. Hasan. FISTEM: A CAD tool for synthesis of easily testable FSM. In *Proc. Fourth Annual IEEE International ASIC Conference and Exhibit*, Rochester, NY, September 1991.

[18] R. Puri and M. M. Hasan. PLASMA: A FSM design kernel. In *Proc. Third Annual IEEE ASIC Seminar and Exhibit*, pages P 16.2.1.-16.2.4., Rochester, NY, September 1990.

[19] T. S. Raghuram. Fault modelling and testing of programmable logic arrays. Master's thesis, IIT, Kanpur, February, 1989.

[20] T. S. Raghuram and M. M. Hasan. PLATEST: A PLA test generator. In *Proc. Fourth CSI/ IEEE International Symposium on VLSI Design*, pages 288-289, New Delhi, January, 1991.

[21] G. Saucier, M. C. Depaulet, P. Sicard. ASYL: A rule-based system for controller synthesis. *IEEE Transactions on Computer-Aided Design*, CAD6(6):1088-1098, 1987.

[22] H. G. Schwaertzel. Testing of VLSI-circuits. In F. Anceau and E. J. Aas, editors, *VLSI Design of Digital Systems*, pages 21-23, Amsterdam, 1983. Elsevier Science Publishers.

[23] J. E. Smith. Detections of faults in programmable logic arrays. *IEEE Transactions on computers*, C-28:845-853, 1979.

[24] F. Somenzi, S. Gai, M. Mezzalama, and P. Prinetto. PART: Programmable array testing based on a partitioning algorithm. *IEEE Transactions on Computer-Aided Design*, CAD-3:142-149, 1984.

[25] H.-K. Tony Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. Test generation for sequential circuits. *IEEE Transactions on Computer-Aided Design*, 7(10):1081-1093, 1988.

[26] R. S. Wei and A. Sangiovanni-Vincentelli. PLATYPUS: A PLA test pattern generation tool. *IEEE Transactions on Computer-Aided Design*, CAD-5(4):633-644, 1986.

[27] T. W. Williams. Trends in design for testability. In F. Lombardi and M. Sami, editors, *Proc. NATO ASI on Testing and Diagnosis of VLSI and ULSI*, pages 1-31, Como, Italy, 1987. Kluwer Academic Publishers.